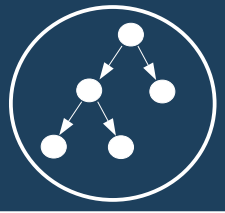


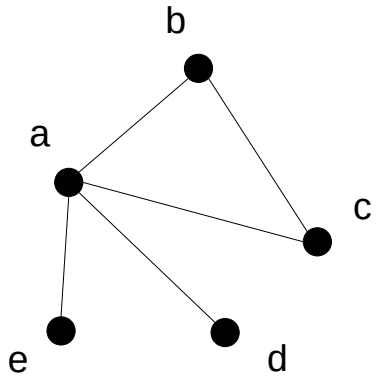
Data Structures and Algorithms

Структуры данных.
Способы представления простого графа

Простой граф

Граф в котором отсутствуют петли называется простым графом.

$$G(V,E), V \neq \emptyset, E \subseteq V \times V, \{v,v\} \notin E, v \in V$$



Распространенные способы представления простого графа

Для создания структуры данных с помощью которой можно представить простой граф в информатике чаще всего используется несколько подходов:

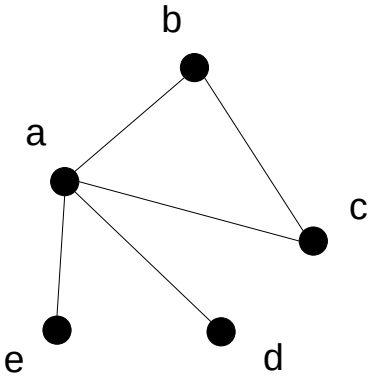
- Массив (список) ребер
- Массив (список) смежности
- Матрица смежности

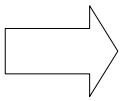
Вне зависимости от представления должны поддерживаться следующие базовые операции:

- Добавление и удаление вершины
- Добавление и удаление ребра
- Проверка на смежность вершин
- Получение всех смежных вершин для данной
- Получение данных хранимых вершиной
- Установка новых данных хранимых в вершине

Массив (список) ребер

Массив ребер. Используется массив или список, который хранит пары смежных вершин. Таким образом наличие такой пары означает, что две вершины связаны.

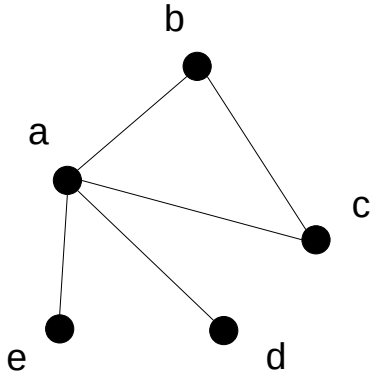


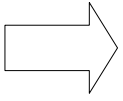


[[a,e], [a,d], [a,c], [a,b], [b,c]]

Массив (список) ребер

Для упрощения реализации можно отдельно хранить массив вершин и в списке ребер хранить ссылки на соответствующую вершину. В таком случае значительно упроститься добавление вершин в граф (достаточно просто добавить ее в массив) и также просто реализуется добавление и удаление ребра (достаточно просто добавить или удалить пару из массива ребер). В тоже время такая реализация плохо подходит для определения есть ли связь между двумя узлами (нужно перебрать все элементы массива ребер) и довольно затратна по памяти.





[a, b, c, d, e]

[[a,e], [a,d], [a,c], [a,b], [b,c]]

Возможное представление вершины графа

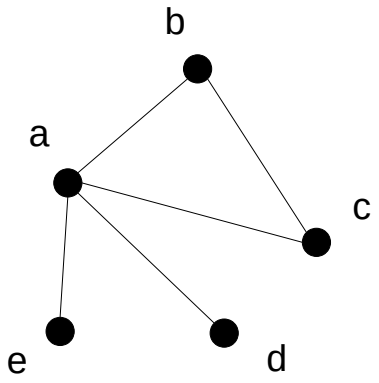
Пожалуй одним из самых простых представлений вершины графа является структура которая хранит однозначный идентификатор вершины (порядковый номер, уникальное имя), и поле для хранения данных связанных с этой вершиной.

Node
id
data

Поле для хранения данных вершины

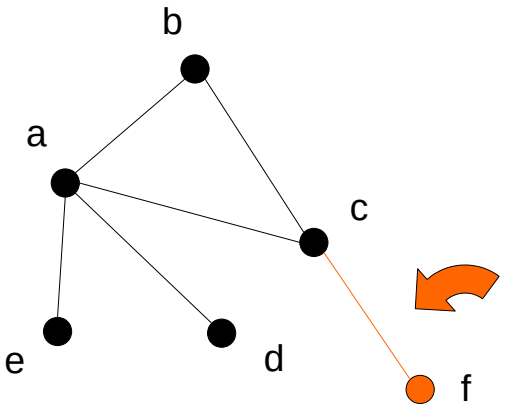
В таком случае операции получения и установки данных связанных с этой вершиной сведутся к чтению и установки поля для хранения данных этой вершины.

Добавление вершины и ребра



[a, b, c, d, e]

[[a,e], [a,d], [a,c], [a,b], [b,c]]



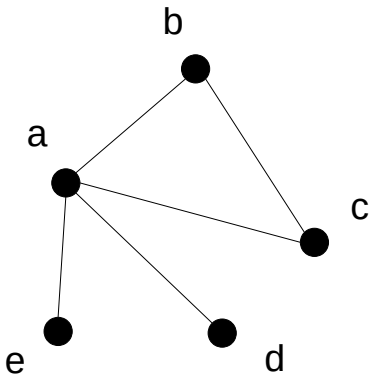
Добавление с список вершин

[a, b, c, d, e, f]

Добавление ребра

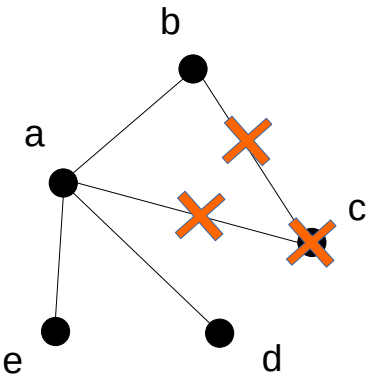
[[a,e], [a,d], [a,c], [a,b], [b,c], [c,f]]

Удаление вершины



[a, b, c, d, e]

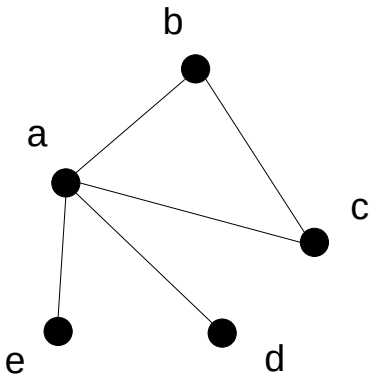
[[a,e], [a,d], [a,c], [a,b], [b,c]]



[a, b, ~~c~~, d, e] Удаляем вершину из списка вершин

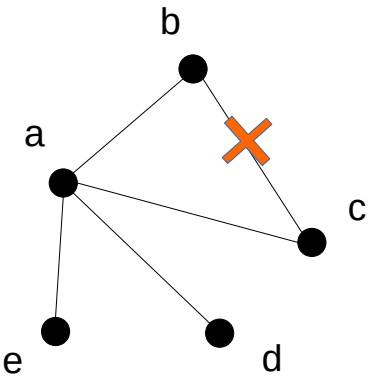
[[a,e], [a,d], [~~a,c~~], [a,b], [~~b,c~~]] Удаляем ребра с ней

Удаление ребра



[a, b, c, d, e]

[[a,e], [a,d], [a,c], [a,b], [b,c]]



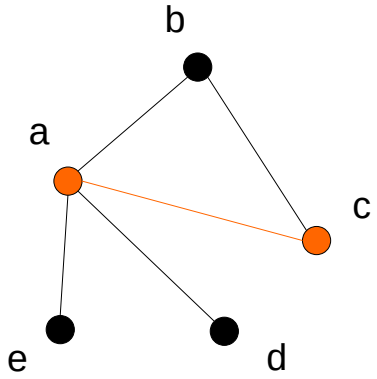
[a, b, c, d, e]

[[a,e], [a,d], [a,c], [a,b], [b,c]]

Удаляем нужное ребро

Определение смежности вершин

Для определения смежности двух вершин достаточно перебрать массив (список) ребер. Если вершины смежные, то есть ребро их связывающее (по определению). Недостатком такого подхода является необходимость полного перебора списка ребер.



Пример: смежны ли вершины a и c. Ответ да, так как есть такое ребро в списке ребер.

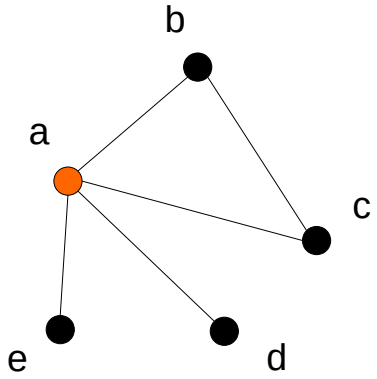
[[a,e], [a,d], [a,c], [a,b], [b,c]]

[a, b, c, d, e]

[[a,e], [a,d], [a,c], [a,b], [b,c]]

Поиск вершин смежных данной

Для поиска всех вершин смежных данной нужно выделить все ребра (в которых встречается данная вершина) и собрать множество вершин являющихся инцидентными данному ребру кроме данной.



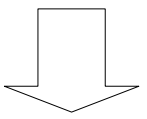
[a, b, c, d, e]

[[a,e], [a,d], [a,c], [a,b], [b,c]]

Пример: получить список вершин смежных вершине a.

[a, b, c, d, e]

[[a,e], [a,d], [a,c], [a,b], [b,c]]



[b, c, d, e]

Реализация на Python

Описание структуры вершины и графа

```
class Graph:

    class Node:
        def __init__(self, id, data):
            self.id = id
            self.data = data

    def __init__(self):
        self.node_list = []
        self.edge_list = []
```

Метод поиска вершины по индификатору

```
def find_node_by_id(self, id):
    for node in self.node_list:
        if node.id == id:
            return node
    return None
```

Методы добавления и удаления вершины

```
def add_node(self, node_id, node_data=None):
    if self.find_node_by_id(node_id) is not None:
        raise Exception("Node with this id already exists")
    new_node = Graph.Node(node_id, node_data)
    self.node_list.append(new_node)

def remove_node(self, node_id):
    remove_node = self.find_node_by_id(node_id)
    if remove_node is None:
        raise Exception("No node with this id")
    remove_edge_list = []
    for edge in self.edge_list:
        if edge[0] == remove_node or edge[1] == remove_node:
            remove_edge_list.append(edge)
    for edge in remove_edge_list:
        self.edge_list.remove(edge)
    self.node_list.remove(remove_node)
```

Методы добавления и удаления ребра

```
def add_edge(self, node_id_a, node_id_b):
    node_from = self.find_node_by_id(node_id_a)
    node_to = self.find_node_by_id(node_id_b)
    if node_from is None or node_to is None:
        raise Exception("No node with this id")
    if node_from == node_to:
        raise Exception("Loop edge")
    self.edge_list.append((node_from, node_to))

def remove_edge(self, node_id_a, node_id_b):
    node_from = self.find_node_by_id(node_id_a)
    node_to = self.find_node_by_id(node_id_b)
    if node_from is None or node_to is None:
        raise Exception("No node with this id")
    for i in range(len(self.edge_list)):
        edge = self.edge_list[i]
        if edge[0] == node_from and edge[1] == node_to:
            del (self.edge_list[i])
            return True
        elif edge[1] == node_from and edge[0] == node_to:
            del (self.edge_list[i])
            return True
    return False
```

Метод проверки смежности вершин

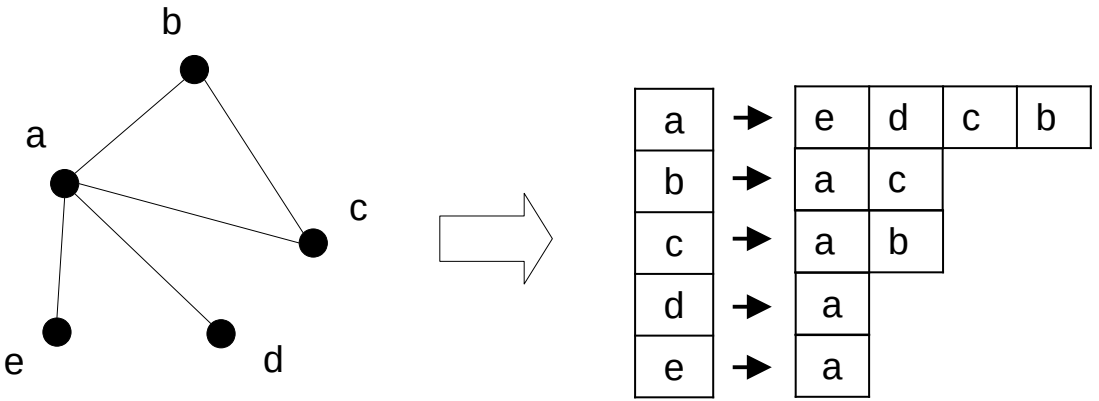
```
def adjacent(self, node_id_a, node_id_b):
    node_from = self.find_node_by_id(node_id_a)
    node_to = self.find_node_by_id(node_id_b)
    if node_from is None or node_to is None:
        raise Exception("No node with this id")
    for edge in self.edge_list:
        if edge[0] == node_from and edge[1] == node_to:
            return True
        if edge[1] == node_from and edge[0] == node_to:
            return True
    return False
```

Метод поиска вершин смежных данной

```
def neighbors(self, node_id):
    node_from = self.find_node_by_id(node_id)
    if node_from is None:
        raise Exception("No node with this id")
    neighbors_node_id = set()
    for edge in self.edge_list:
        if edge[0] == node_from:
            neighbors_node_id.add(edge[1].id)
        if edge[1] == node_from:
            neighbors_node_id.add(edge[0].id)
    return neighbors_node_id
```

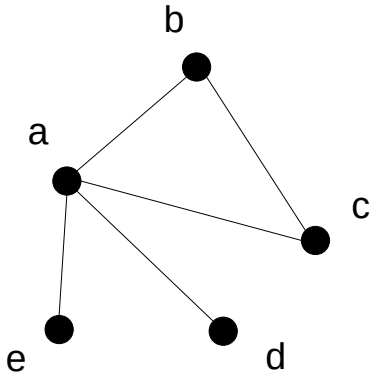
Представление в виде списков смежности

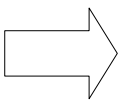
Представление графа в виде списка смежности подразумевает хранение списка вершин, в тоже время сама вершина хранит список ссылок на смежные к ней вершины. Такой алгоритм довольно оптимален для определения смежности двух вершин и поиска всех вершин смежных данной.

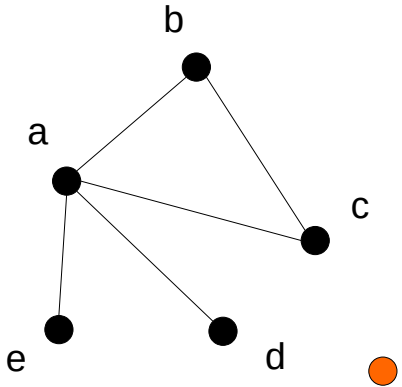


Добавление вершины

Для добавления вершины достаточно просто добавить ее в список вершин.







Добавление вершины

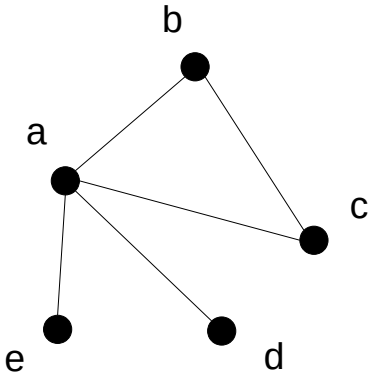
a	e d c b
b	a c
c	a b
d	a
e	a

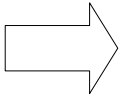
Добавление вершины

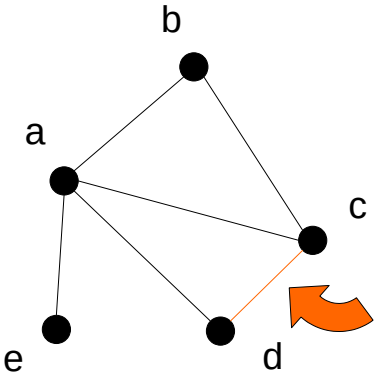
a	e d c b
b	a c
c	a b
d	a
e	a
f	

Добавление ребра

Для добавления ребра нужно найти инцидентные ему вершины. Для найденных вершин в списке смежных вершин добавить вершину с противоположной стороны ребра.







Добавление ребра

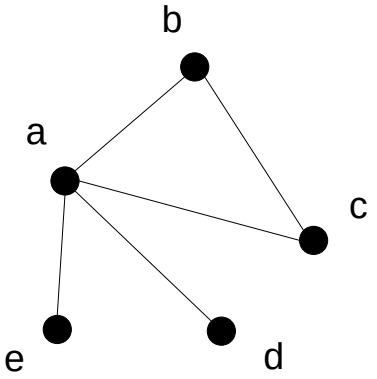
a	→	e	d	c	b
b	→	a	c		
c	→	a	b		
d	→	a			
e	→	a			

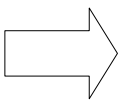
a	→	e	d	c	b
b	→	a	c		
c	→	a	b	d	
d	→	a	c		
e	→	a			

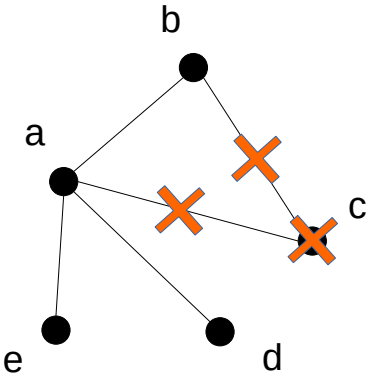
Добавление ребра

Удаление вершины

При удалении вершины нужно перейти к смежным вершинам (ссылки на них есть в списке смежных вершин). Для каждой смежной вершины в списке вершин удалить ссылку на удаляемую. После этого удалять саму вершину.





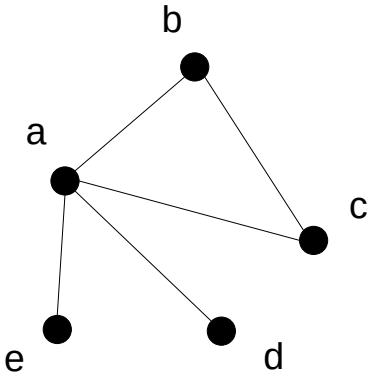


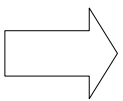
a	→	e	d	c	b
b	→	a	c		
c	→	a	b		
d	→	a			
e	→	a			

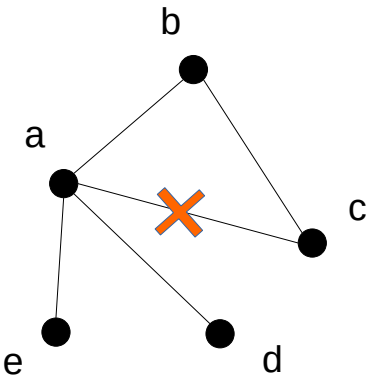
a	→	e	d	c	b
b	→	a	c		
c	→	a	b		
d	→	a			
e	→	a			

Удаление ребра

При удалении ребра нужно перейти к вершинам на концах ребра. Для каждой вершины в списке вершин удалить ссылку на удаляемую.





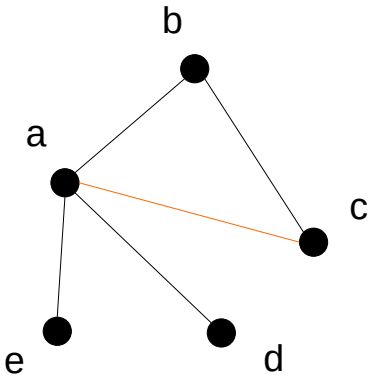


a	→	e	d	c	b
b	→	a	c		
c	→	a	b		
d	→	a			
e	→	a			

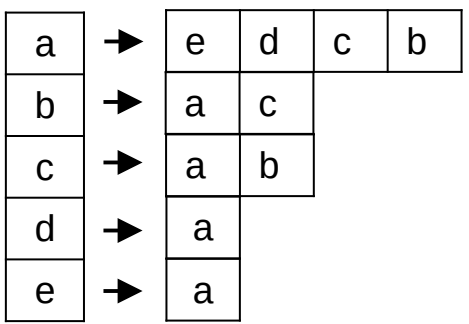
a	→	e	d		b
b	→	a	c		
c	→		b		
d	→	a			
e	→	a			

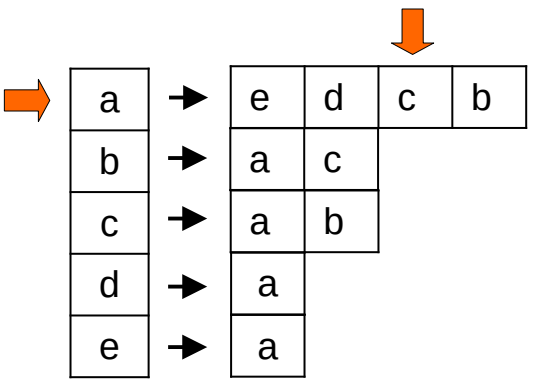
Проверка смежности вершин

Проверка смежности вершин выполняется очевидным образом — переходим к одной из вершин ребра и ищем в ее списке смежных вершин вторую вершину ребра. Если она есть то эти вершины смежные.



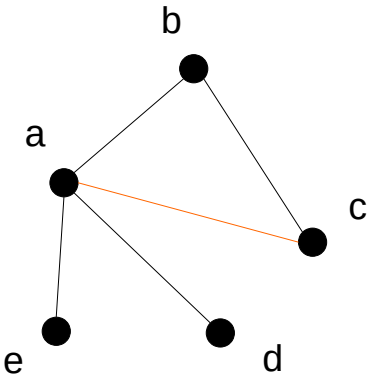
Пример: смежны ли вершины a и c. Переходим к вершине a и в списке смежных с ней вершин находим вершину c. Они смежные.





Получение всех смежных вершин

Для получения всех вершин смежных данной достаточно просто вернуть ее список смежных вершин.



a	→	e	d	c	b
b	→	a	c		
c	→	a	b		
d	→	a			
e	→	a			

Пример: получить список вершин смежных вершине a.

a	→	e	d	c	b
b	→	a	c		
c	→	a	b		
d	→	a			
e	→	a			

Возможное представление вершины графа

Для этого представления графа для описания вершины графа используется структура которая хранит однозначный идентификатор вершины (порядковый номер, уникальное имя), и поле для хранения данных связанных с этой вершиной и список ссылок на смежные с ней вершины.

```
Node
    id
    data
    Node[] nodes
```

Поле для хранения данных вершины

Список ссылок на смежные вершины

Использование разных структур данных для описания графа

Для описания структуры данных представляющей такой граф, можно использовать несколько подходов. Первый это действительно список вершин, второй это ассоциативный массив вершин. Во втором случае возможны два варианта, если вершины хешируемые, то сама вершина выступает ключем, а значение это список смежных с ней вершин. Если вершина не хешируема, то тогда ключ это уникальный идентификатор, значение - вершина которая содержит список смежных вершин.

Реализация на Java

Описание представления графа

```
class Graph {

    private class Node {
        final String id;
        Object data;
        List<Node> adjacentNodes = new ArrayList<>();

        public Node(String id) {
            super();
            this.id = id;
        }
    }

    private final Map<String, Node> nodes = new HashMap<>();
```

Методы добавления вершины

```
public void addNode(String id, Object data) {
    if (nodes.get(id) != null) {
        throw new IllegalArgumentException("Node with this ID already exists");
    }
    Node newNode = new Node(id);
    newNode.data = data;
    nodes.put(id, newNode);
}

public void addNode(String id) {
    addNode(id, null);
}
```

Метод добавления ребра

```
public void addEdge(String idFrom, String idTo) {
    Node nodeFrom = nodes.get(idFrom);
    Node nodeTo = nodes.get(idTo);
    if (nodeFrom == null || nodeTo == null) {
        throw new IllegalArgumentException("Node with this id does not exist");
    }
    if (nodeFrom == nodeTo) {
        throw new IllegalArgumentException("Loop edge");
    }
    nodeFrom.adjacentNodes.add(nodeTo);
    nodeTo.adjacentNodes.add(nodeFrom);
}
```

Метод удаления вершины

```
public void removeNode(String id) {
    Node removeNode = nodes.get(id);
    if (removeNode == null) {
        return;
    }
    for (Node node : removeNode.adjacentNodes) {
        node.adjacentNodes.remove(removeNode);
    }
    nodes.remove(id);
}
```

Метод удаления ребра

```
public void removeEdge(String idFrom, String idTo) {
    Node nodeFrom = nodes.get(idFrom);
    Node nodeTo = nodes.get(idTo);
    if (nodeFrom == null || nodeTo == null) {
        throw new IllegalArgumentException("Node with this id does not exist");
    }
    nodeFrom.adjacentNodes.remove(nodeTo);
    nodeTo.adjacentNodes.remove(nodeFrom);
}
```

Метод проверки смежности вершин

```
public boolean isAdjacent(String idFrom, String idTo) {
     Node nodeFrom = nodes.get(idFrom);
     Node nodeTo = nodes.get(idTo);
     if (nodeFrom == null || nodeTo == null) {
          return false;
     }
     return nodeFrom.adjacentNodes.contains(nodeTo);
}
```

Метод получения списка смежных вершин

```
public String[] getAdjacentNodesId(String id) {
    Node node = nodes.get(id);
    if (node == null) {
        return null;
    }
    String[] ids = new String[node.adjacentNodes.size()];
    for (int i = 0; i < ids.length; i++) {
        ids[i] = node.adjacentNodes.get(i).id;
    }
    return ids;
}
```

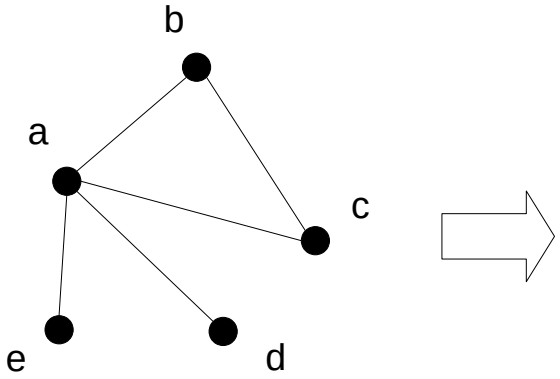
Метод получения данных и установки данных для вершины

```
public Object getNodeDataById(String id) {
    Node node = nodes.get(id);
    if (node == null) {
        return null;
    }
    return node.data;
}

public void setNodeDataById(String id, Object data) {
    Node node = nodes.get(id);
    if (node == null) {
        return;
    }
    node.data = data;
}
```

Матрица смежности

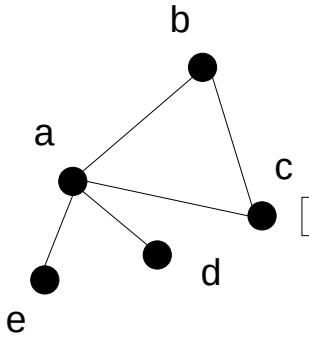
Представление в виде матрицы смежности подразумевает использование двумерного массива количество строк и столбцов в котором равно количеству вершин. Элементом такого массива будет 0 если вершины представляющие строку и столбец не смежные, и положительное число в случае если вершины смежные.



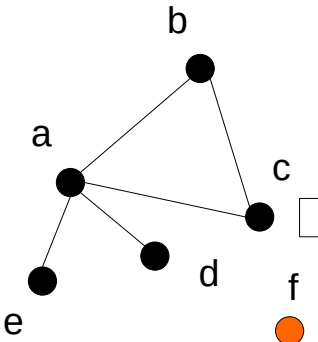
	a	b	c	d	e
a	0	1	1	1	1
b	1	0	1	0	0
c	1	1	0	0	0
d	1	0	0	0	0
e	1	0	0	0	0

Добавление вершины

Добавление вершины в таком представлении одна из самых затратных операций. При этом нужно увеличить размер матрицы на одну единицу и скопировать данные из предыдущей матрицы в текущую.



	a	b	c	d	e
a	0	1	1	1	1
b	1	0	1	0	0
c	1	1	0	0	0
d	1	0	0	0	0
e	1	0	0	0	0



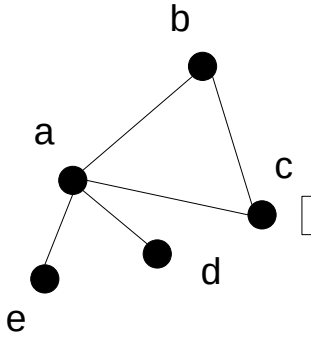
Добавление вершины

	a	b	c	d	e	f
a	0	1	1	1	1	0
b	1	0	1	0	0	0
c	1	1	0	0	0	0
d	1	0	0	0	0	0
e	1	0	0	0	0	0
f	0	0	0	0	0	0

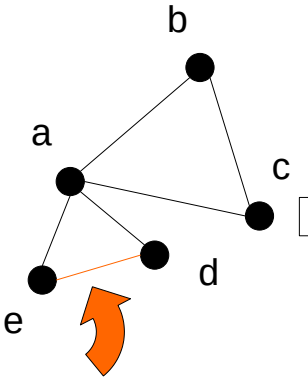
Добавление вершины

Добавление ребра

Добавление ребра выполняется очень быстро (в этом преимущество этого представления). На пересечении строки и столбца отвечающих за смежные вершины для этого ребра поставить значение больше 0.



	a	b	c	d	e
a	0	1	1	1	1
b	1	0	1	0	0
c	1	1	0	0	0
d	1	0	0	0	0
e	1	0	0	0	0



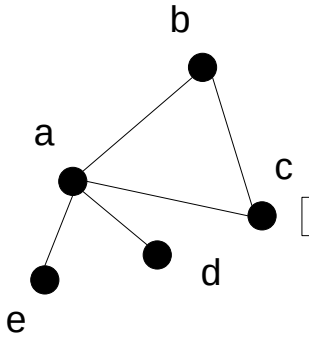
Добавление ребра

	a	b	c	d	e
a	0	1	1	1	1
b	1	0	1	0	0
c	1	1	0	0	0
d	1	0	0	0	1
e	1	0	0	1	0

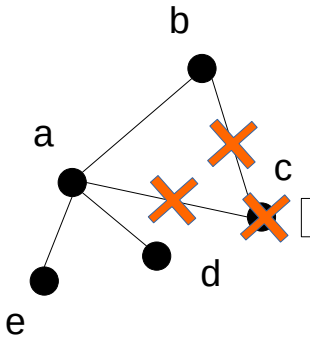
Добавление ребра

Удаление вершины

Для удаления вершины нужно или фактически удалить строку и столбец с заданной вершиной, или пометить его как удаленный (ленивое удаление).



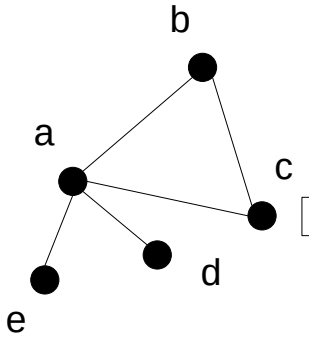
	a	b	c	d	e
a	0	1	1	1	1
b	1	0	1	0	0
c	1	1	0	0	0
d	1	0	0	0	0
e	1	0	0	0	0



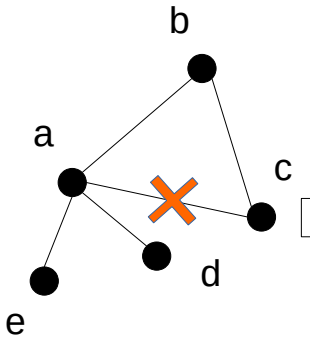
	a	b	c	d	e
a	0	1	1	1	1
b	1	0	1	0	0
c	1	1	0	0	0
d	1	0	0	0	1
e	1	0	0	1	0

Удаление ребра

Для удаления ребра нужно установить значение равное 0 в строке и столбце соответствующие вершинам этого ребра.



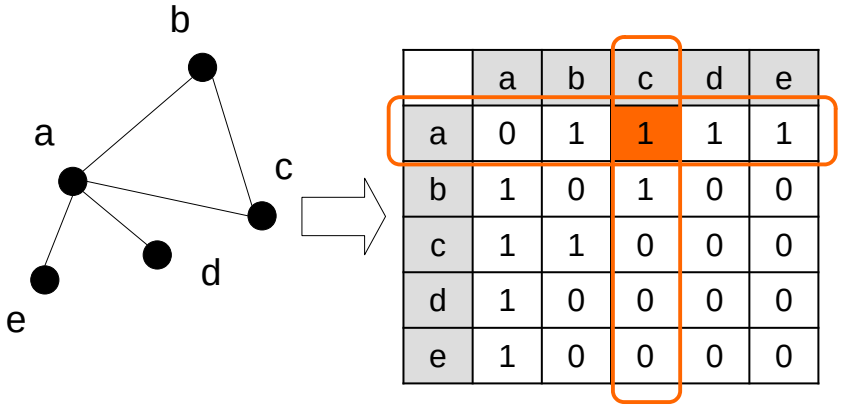
	a	b	c	d	e
a	0	1	1	1	1
b	1	0	1	0	0
c	1	1	0	0	0
d	1	0	0	0	0
e	1	0	0	0	0



	a	b	c	d	e
a	0	1	**0**	1	1
b	1	0	1	0	0
c	**0**	1	0	0	0
d	1	0	0	0	1
e	1	0	0	1	0

Проверка вершин на смежность

Для проверки вершин на смежность достаточно проверить значение стоящее на пересечении строки и столбца соответствующих данным вершинам. Если стоит значение больше нуля то вершины смежные.

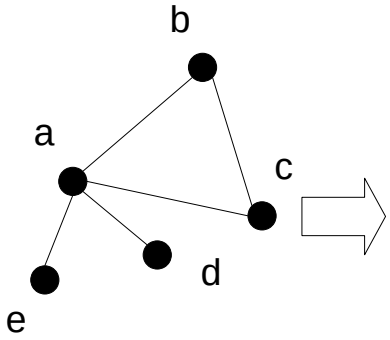


	a	b	c	d	e
a	0	1	1	1	1
b	1	0	1	0	0
c	1	1	0	0	0
d	1	0	0	0	0
e	1	0	0	0	0

Пример: смежны ли вершины **a** и **c**. На пересечении строки (с вершиной **a**) и столбца (с вершиной **c**) стоит 1. Значит они смежные.

Получение вершин смежных данной

Для получения вершин смежных данной следует выполнить следующие действия. Выбрать столбец соответствующий данной вершине. Все вершины для которых в этом столбце стоит значение больше нуля смежные ей.



	a	b	c	d	e
a	0	1	1	1	1
b	1	0	1	0	0
c	1	1	0	0	0
d	1	0	0	0	0
e	1	0	0	0	0

Пример: получить список вершин смежных вершине **c**. Выделяем строку соответствующую этой вершине выбираем вершины напротив которых стоит значение отличное от 0. Это вершины **a** и **b**.

Реализация на Fortran

Описание вершины и графа

```
type Node
    character(len=10)::node_id
    integer::node_data
    logical::is_present
end type Node

type Graph
    type(Node), allocatable::nodes(:)
    integer, allocatable::adjacency_matrix(:,:)
    integer::matrix_size
    contains

    procedure,pass::init
    procedure,pass::add_node
    procedure,pass::add_edge
    procedure, pass::find_node_index_by_id
    procedure, pass::remove_node
    procedure, pass::remove_edge
    procedure, pass::print_graph
    procedure, pass::is_node_adjacency
    procedure, pass::get_adjacency_nodes_index
end type Graph
```

Процедура инициализации и поиска вершины по id

```
subroutine init(this)
    class(Graph)::this
    this%matrix_size = 100
    allocate(this%nodes(this%matrix_size))
    allocate(this%adjacency_matrix(this%matrix_size,this%matrix_size))
    this%nodes%is_present = .false.
    this%adjacency_matrix = 0
end subroutine init

subroutine find_node_index_by_id(this, node_id, node_index)
    class(Graph)::this
    character(len=*), intent(in)::node_id
    integer, intent(inout)::node_index
    integer::i
    node_index = -1
    do i = 1, size(this%nodes)
        if (this%nodes(i)%is_present .and. this%nodes(i)%node_id == node_id) then
            node_index = i
            exit
        end if
    end do
end subroutine find_node_index_by_id
```

Процедура добавления вершины

```
subroutine add_node(this, node_id, node_data, op_result)
    class(Graph)::this
    character(len=*), intent(in)::node_id
    integer, intent(in)::node_data
    logical, intent(inout)::op_result
    integer::i
    op_result = .false.
    call this%find_node_index_by_id(node_id,i)
    if(i /= -1) then
        return
    end if
    do i = 1, size(this%nodes)
        if (.not.this%nodes(i)%is_present) then
            this%nodes(i)%is_present = .true.
            this%nodes(i)%node_id = node_id
            this%nodes(i)%node_data = node_data
            op_result = .true.
            exit
        end if
    end do
end subroutine add_node
```

Процедура добавления ребра

```
subroutine add_edge(this, node_id_from, node_id_to, op_result)
    class(Graph)::this
    character(len=*),intent(in)::node_id_from, node_id_to
    logical, intent(inout)::op_result
    integer::i, j
    i = -1
    j = -1
    op_result = .false.
    call this%find_node_index_by_id(node_id_from, i)
    call this%find_node_index_by_id(node_id_to, j)
    if(i == -1 .or. j == -1) then
        return
    end if
    this%adjacency_matrix(i,j) = 1
    this%adjacency_matrix(j,i) = 1
    op_result = .true.
end subroutine add_edge
```

Процедура удаления вершины

```
subroutine remove_node(this, node_id, op_result)
    class(Graph)::this
    character(len=*),intent(in)::node_id
    logical, intent(inout)::op_result
    integer::i
    i = -1
    op_result = .false.
    call this%find_node_index_by_id(node_id, i)
    if (i == -1) then
        return
    end if
    this%nodes(i)%is_present = .false.
    this%adjacency_matrix(i,:) = 0
    this%adjacency_matrix(:,i) = 0
    op_result = .true.
end subroutine remove_node
```

Процедура удаления вершины

```
subroutine remove_edge(this, node_id_from, node_id_to, op_result)
    class(Graph)::this
    character(len=*), intent(in)::node_id_from, node_id_to
    logical, intent(inout)::op_result
    integer::i, j
    i = -1
    j = -1
    op_result = .false.
    call this%find_node_index_by_id(node_id_from, i)
    call this%find_node_index_by_id(node_id_to, j)
    if(i == -1 .or. j == -1) then
        return
    end if
    this%adjacency_matrix(i,j) = 0
    this%adjacency_matrix(j,i) = 0
    op_result = .true.
end subroutine remove_edge
```

Процедура проверки вершин на смежность

```
subroutine is_node_adjacency(this, node_id_from, node_id_to, op_result)
    class(Graph)::this
    character(len=*), intent(in)::node_id_from, node_id_to
    logical, intent(inout)::op_result
    integer::i, j
    i = -1
    j = -1
    op_result = .false.
    call this%find_node_index_by_id(node_id_from, i)
    call this%find_node_index_by_id(node_id_to, j)
    if(i == -1 .or. j == -1) then
        return
    end if
    if (this%adjacency_matrix(i,j)/=0) then
        op_result = .true.
    end if
end subroutine is_node_adjacency
```

Эффективность предложенных представлений

	Занимаемая память	Добавление вершины	Добавление ребра	Удаление вершины	Удаление ребра	Проверка на смежность вершин	Получение смежных вершин
Список ребер	O(E)	O(1)	O(1)	O(E) + O(V)	O(E)	O(E)	O(E)
Список смежности	O(E) + O(V)	O(1)	O(1)	O(E)	O(V)	O(V)	O(V)
Матрица смежности	$O(V^2)$	$O(V^2)$	O(1)	$O(V^2)$	O(1)	O(1)	O(V)

Представление в виде списков смежности стоит применять для разреженных графов (число ребер гораздо меньше квадрата количества вершин). Представление в виде матрица смежности стоит использовать для плотных графов (количество ребер сопоставимо с квадратом количества вершин).

Список литературы

1) Джеймс А. Андерсон «Дискретная математика и комбинаторика». Издательский дом «Вильямс», 2004.
2) Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн // Алгоритмы: построение и анализ 3-е издание. — М.: «Вильямс», 2013. — С. 1328. ISBN 978-5-8459-1794-2
3) Роберт Седжвик, Кевин Уэйн «Алгоритмы на java 4-е издание» Пер. с англ. - М. : ООО "И.Д. Вильямс", 2013. ISBN 978-5-8459-1781-2.