Лекция по курсу «Алгоритмы и структуры данных» / «Технологии и методы программирования»

Графы:

основные определения, представление графов в памяти ЭВМ, алгоритмы обхода графов

Мясников Е.В.

Графы: Основные определения

Граф или неориентированный граф *G* — это упорядоченная пара *G* = <*V*, *E*>, где *V* называется множеством вершин или узлов, а E – множеством ребер.

Каждое ребро $e \in E$ задается парой вершин (*u*,*v*), которые оно соединяет. Вершины *u* и *v* называются ***концевыми*** вершинами (концами) ребра.

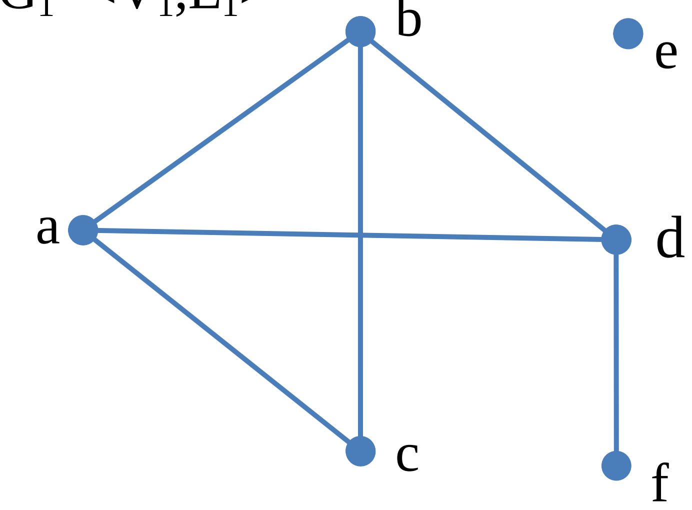
Две концевые вершины одного и того же ребра называются ***соседними***.

Мы будем иметь дело с конечными графами, в которых множества *V* и *E* конечны.

Вершины и рёбра графа называются также ***элементами*** графа,
число вершин в графе |*V*| — ***порядком***,
а число рёбер |*E*|— ***размером*** графа.

Два ребра называются ***смежными***, если они имеют общую концевую вершину.

G_1=<V_1,E_1>



$V_1 = \{ a, b, c, d, e, f \};$

$E_1 = \{ (a,b), (a,c), (a,d),$
$(b,c), (b,d), (d,f) \}$

Графы: Основные определения

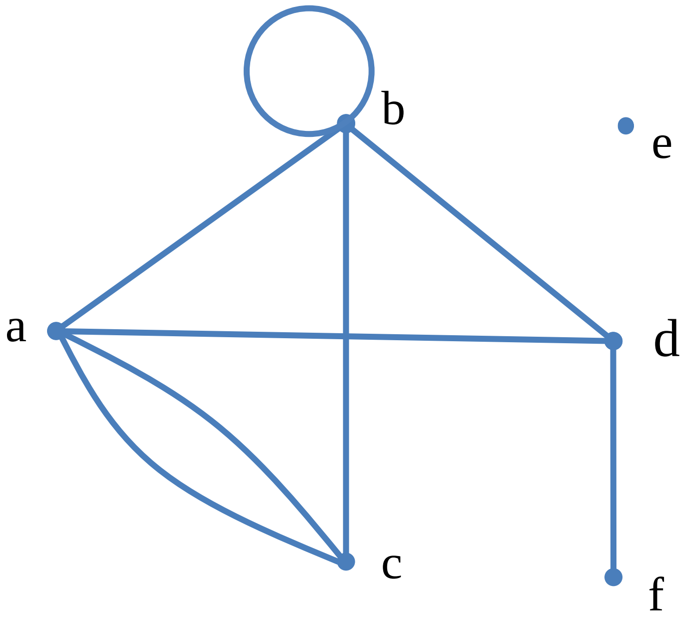
Два ребра называются ***кратными***, если множества их концевых вершин совпадают.

Ребро называется ***петлёй***, если его концы совпадают, то есть $e = (v,v)$.

Степенью вершины v называют количество рёбер, для которых она является концевой (петли считают дважды).

Вершина называется ***изолированной***, если она не является концом ни для одного ребра.

Вершина называется **висячей** (или **листом**), если она является концом ровно одного ребра.



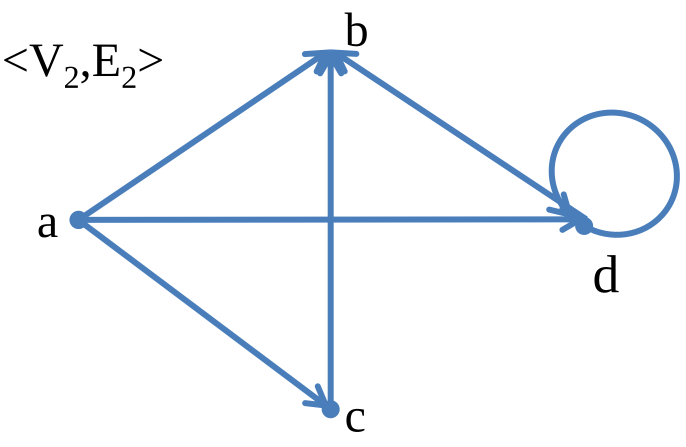
Графы: Основные определения

В ***ориентированном*** графе (орграфе) элементами множества ребер *E* являются упорядоченные пары вершин, называемые дугами или ориентированными рёбрами.

В дуге (*u, v*), вершину *u* называют началом, а *v* — концом дуги. Дуга *u v* ведёт от вершины *u* к вершине *v*.

В ***смешанном*** графе некоторые рёбра могут быть ориентированными, а некоторые — неориентированными (ориентированный и неориентированный графы являются частными случаями смешанного).

G_2=<V_2,E_2>



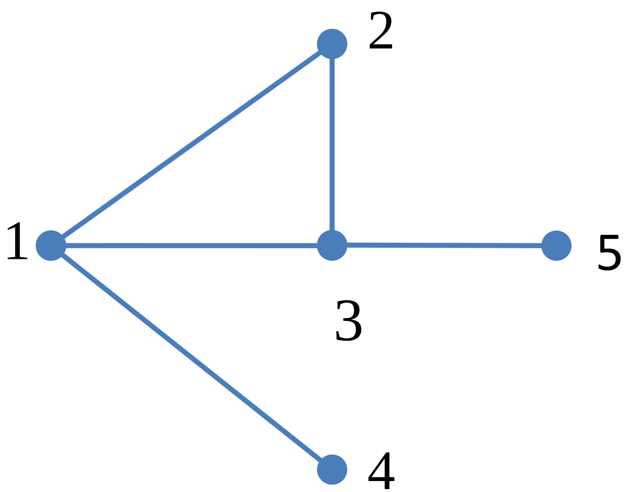
V_2 = { a, b, c, d };
E_2 = { (a,b), (a,c), (a,d),
(c,b), (b,d), (d,d) }

Способы представления графов в памяти ЭВМ

Матрица смежности - таблица, где как столбцы, так и строки соответствуют вершинам графа.

В каждой ячейке этой матрицы записывается число, определяющее наличие связи от вершины-строки к вершине-столбцу (либо наоборот).

Недостатком является требования к памяти - очевидно, квадрат количества вершин.



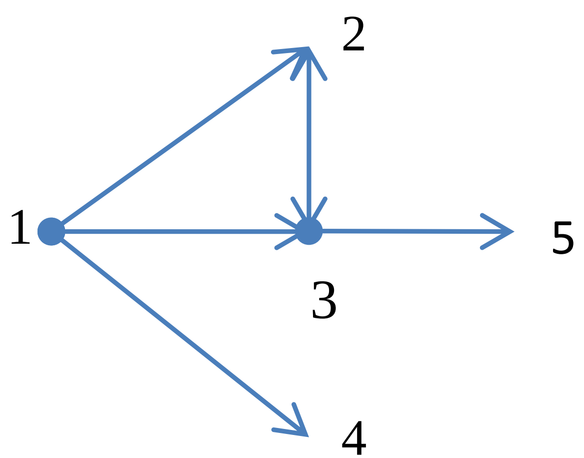
	1	2	3	4	5
1	0	1	1	1	0
2	1	0	1	0	0
3	1	1	0	0	1
4	1	0	0	0	0
5	0	0	1	0	0

Представление графа в виде матрицы смежности

Способы представления графов в памяти ЭВМ

Матрица инцидентности **–** таблица, в которой строки соответствуют вершинам графа, а столбцы – ребрам графа.

В ячейку на пересечении *i*-ой строки с *j*-м столбцом матрицы записывается 1 в случае если ребро *j* выходит из вершины *i*, -1 если ребро входит в указанную вершину, любое число отличное от 0,1,-1 если связь является петлей, и 0 во всех остальных случаях.



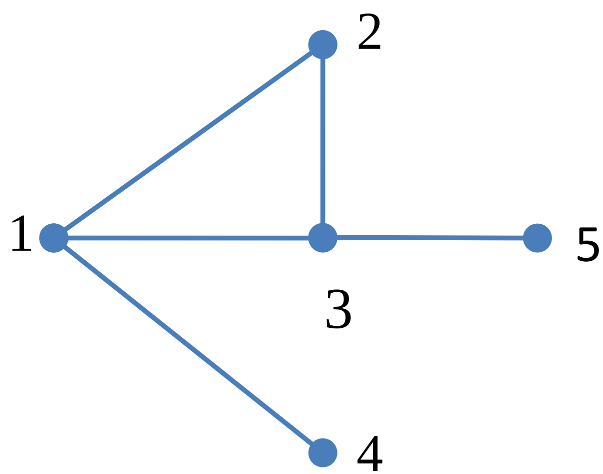
	1	2	3	4	5	6
1	1	1	1	0	0	0
2	-1	0	0	1	-1	0
3	0	-1	0	-1	1	1
4	0	0	-1	0	0	0
5	0	0	0	0	0	-1

Представление графа в виде матрицы инцидентности для ***ориентированного*** графа

Способы представления графов в памяти ЭВМ

Матрица инцидентности – таблица, в которой строки соответствуют вершинам графа, а столбцы – ребрам графа.

В ячейку на пересечении *i*-ой строки с *j*-м столбцом матрицы записывается 1 в случае если ребро *j* выходит из вершины *i*, -1 если ребро входит в указанную вершину, любое число отличное от 0,1,-1 если связь является петлей, и 0 во всех остальных случаях.



	1	2	3	4	5
1	1	1	1	0	0
2	1	0	0	1	0
3	0	1	0	1	1
4	0	0	1	0	0
5	0	0	0	0	1

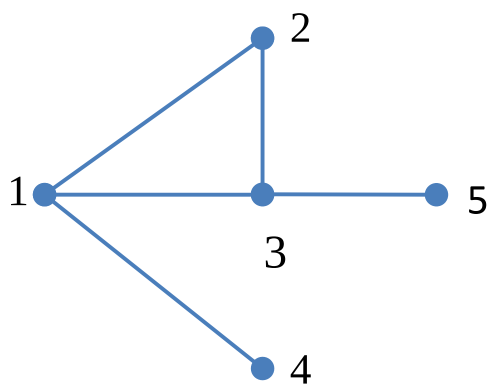
Представление графа в виде матрицы инцидентности
для ***неориентированного*** графа

Данный способ является самым емким (размер пропорционален $|E|\ |V|$) и неудобным для хранения, но облегчает нахождение циклов в графе.

Способы представления графов в памяти ЭВМ

Список рёбер —тип представления графа в памяти, подразумевающий, что каждое ребро представляется двумя числами — номерами вершин этого ребра.

Список рёбер более удобен для реализации различных алгоритмов на графах по сравнению с матрицей смежности.

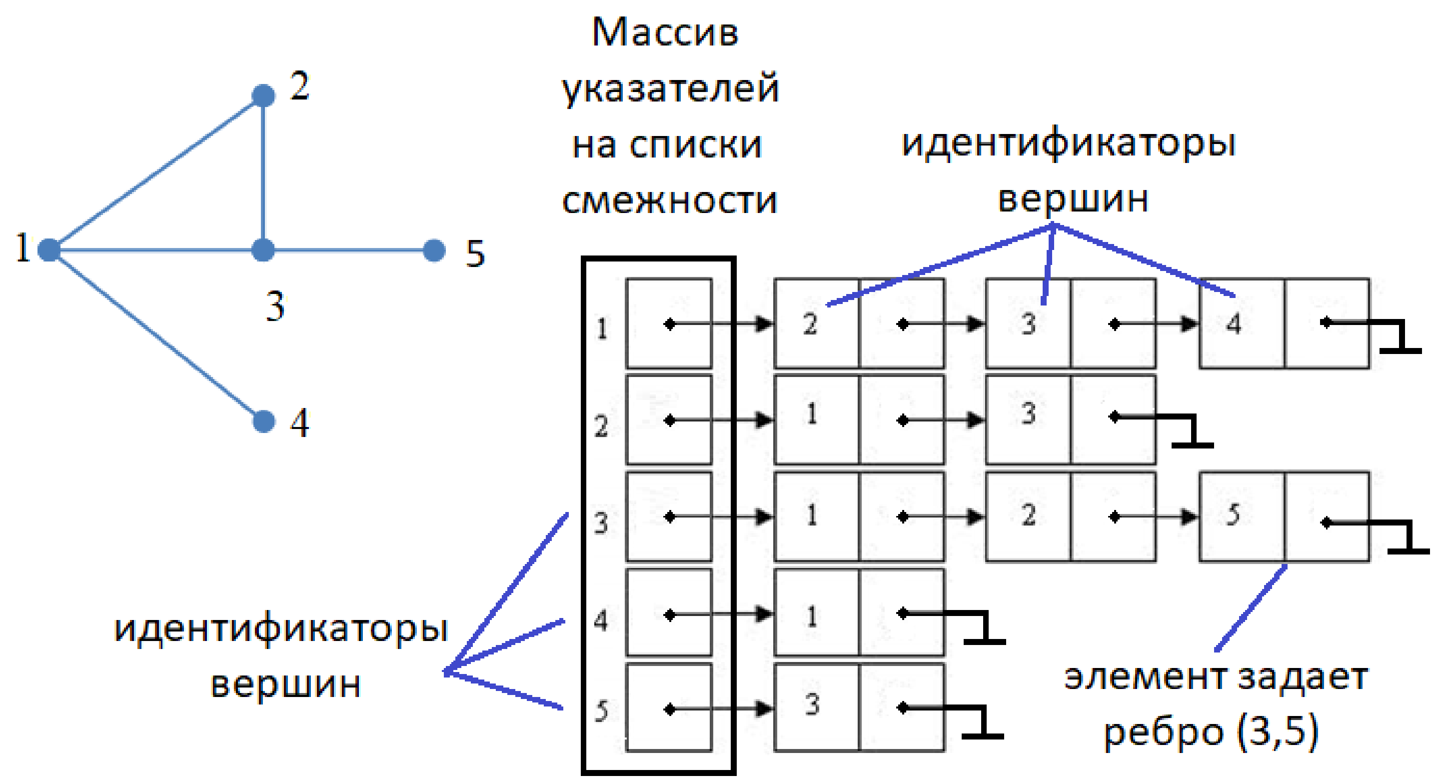


1	1	1	2	3	2	3	4	3	5
2	3	4	3	5	1	1	1	2	3

Представление графа в виде списка ребер

Способы представления графов в памяти ЭВМ

Списки смежности - способ представления графа с использованием динамических структур данных – списков. При использовании этого способа хранят одномерный массив (или список) элементов, соответствующих вершинам графа. Каждый элемент такого массива содержит необходимую информацию о вершине *u* (идентификатор, сопутствующую информацию), а также указатель на список смежности, представляющий собой линейную последовательность элементов, содержащих идентификаторы v_i тех вершин, в которые ведут ребра (u, v_i), исходящие из вершины *u*.



Алгоритмы обхода графов

Поиск в глубину

Начинаем обход из некоторой начальной вершины *s* и поочередно исследуем все ребра, выходящие из нее. Для обхода нам понадобится информация о том, какие вершины мы уже посещали, чтобы избежать их повторного посещения.

В классическом изложении алгоритмов поиска на графах в качестве такого признака используется ***цвет вершины***, принимающий следующие значения:
белый – вершина еще не посещалась
серый – вершина уже была открыта, но обработка связанных с ней вершин еще не завершена
черный – список смежности вершины исследован, и обработка вершины завершена

Кроме цвета для понимания работы алгоритма полезными являются ***метки времени*** (условные моменты времени, обозначаемые целыми числами:
$t^{откр}$ - время открытия вершины, соответствует окрашиванию вершины в серый
$t^{закр}$ - время закрытия вершины, соответствует окрашиванию в черный цвет

Для сохранения информации о пути следования в каждую конкретную вершину, посещаемую во время обхода, будем использовать:
$Пред_u$ – идентификатор вершины из которой при обходе осуществляется переход в вершину *u*.

Поиск в глубину: инициализация

На этапе инициализации все вершины окрашены в белый цвет, их предшественники не определены, а метки времени не выставлены:

Инициализация():

Для каждой вершины $u \in V$:

Цвет$_u$:= белый

Пред$_u$:= NULL

$\boldsymbol{t_u^{\text{откр}}}$ **:= NULL**

$\boldsymbol{t_u^{\text{закр}}}$ **:= NULL**

t := 0

Поиск в глубину: процедура обхода

Поиск_в_глубину(s):

  Цвет$_s$:= серый

  $t_s^{\text{откр}}$ **:= t**

  t := t + 1

  Для каждой вершины u смежной с s:

    Если (Цвет$_u$ = белый):

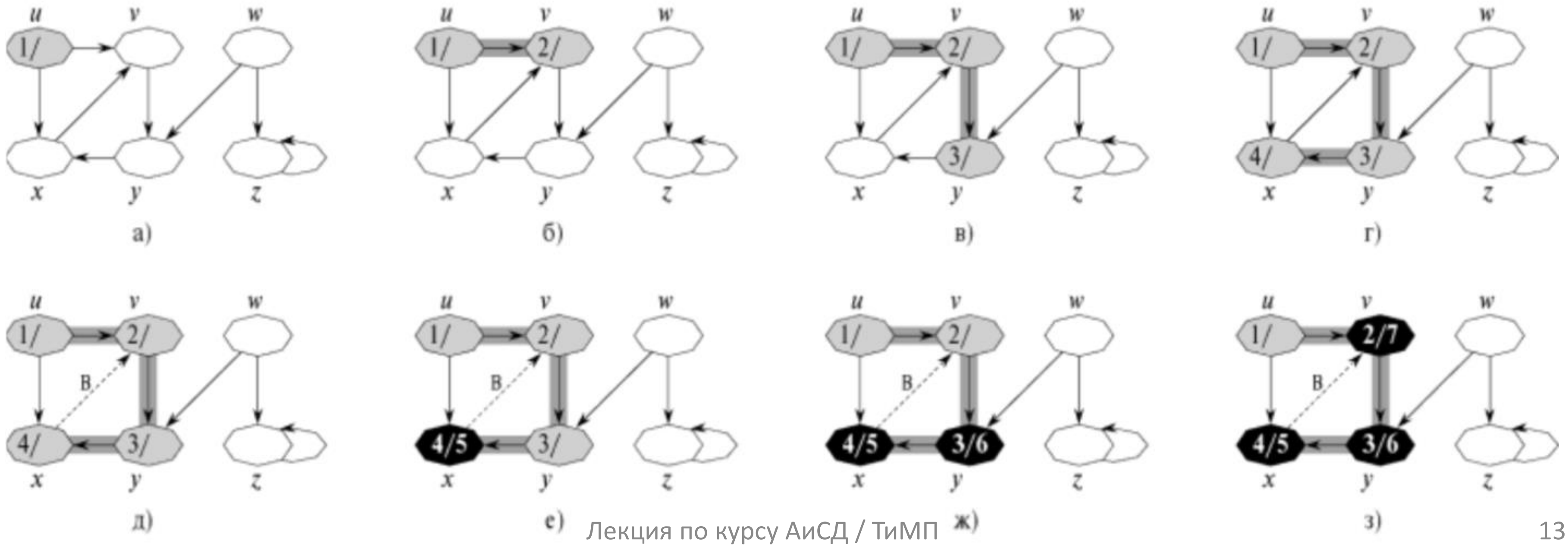
      Пред$_u$:= s

      Поиск_в_глубину(u)

  Цвет$_s$:= черный

  $t_s^{\text{закр}}$ **:= t**

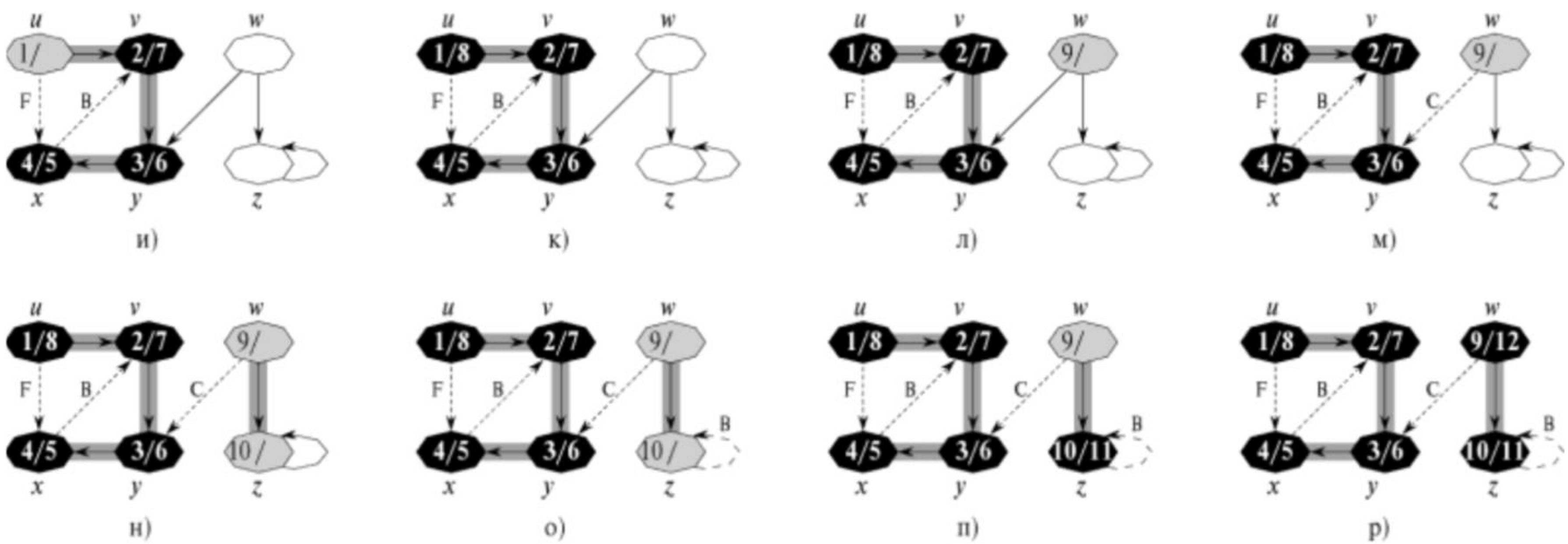
  t := t + 1



Поиск в глубину: процедура обхода

Так как, по ряду причин, весь граф может быть не обработан полностью путем обхода из единственной начальной вершины, процедуру поиска может потребоваться запустить для других необработанных вершин:

Для каждой вершины $s \in V$:
Если (Цвет$_s$ = белый):
Поиск_в_глубину(s)



Поиск в ширину

При поиске в глубину мы на прямом ходе рекурсии «углубляемся» внутрь графа настолько, насколько это возможно, удаляясь от начальной вершины.

На практике часто желаемым свойством является исследование сначала всех доступных вершин, находящихся в одном шаге от начальной, затем – в двух шагах и т.д.

Алгоритмом, обеспечивающим такой порядок обхода, является алгоритм поиска в ширину.

При рассмотрении этого алгоритма, как и ранее, для каждой вершины графа будем использовать **цвет вершин** и **предшественника**, из которого осуществляется переход в заданную вершину.

Вместо меток времени, использованных ранее для каждой вершины, будем хранить **расстояние от исходной вершины**, с которой начинается обход. Это расстояние будет обозначаться как d_u, выражаться в числе ребер и представлять собой *кратчайшее расстояние* от исходной вершины до вершины *u*.

Поиск в ширину: инициализация

Перед выполнением алгоритма обхода выполняется инициализация:

Инициализация():

Для каждой вершины $u \in V$:

Цвет$_u$:= белый

Пред$_u$:= NULL

d_u := ∞

Поиск в ширину: процедура обхода

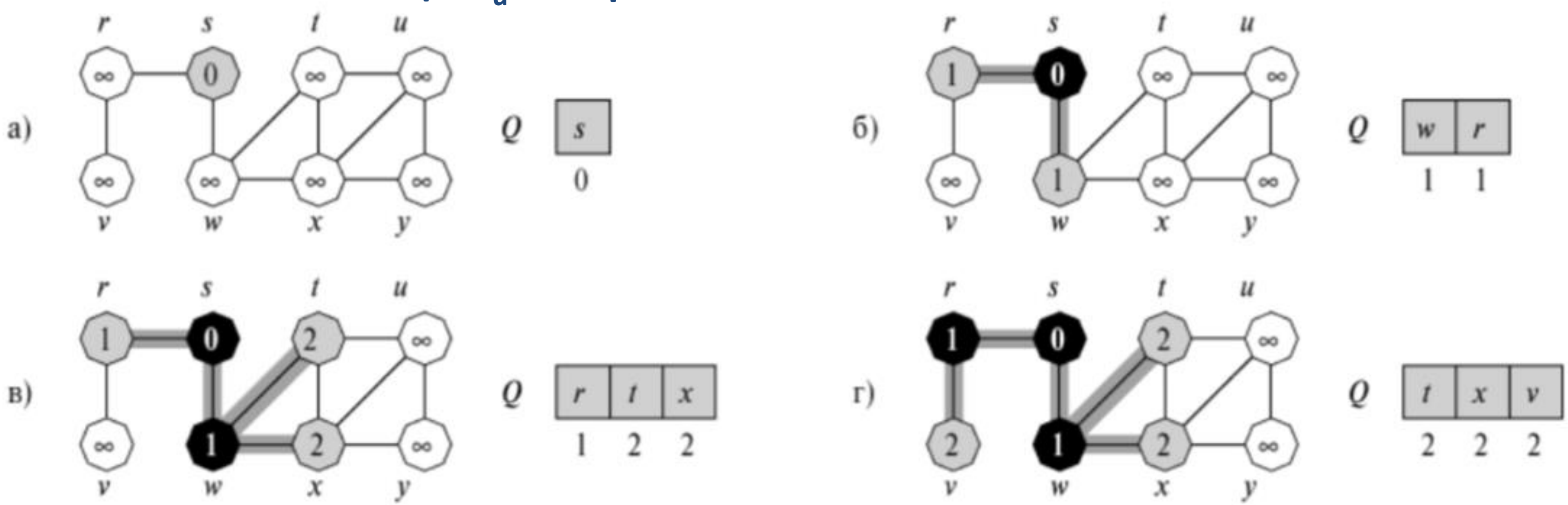
```
Поиск_в_ширину(s):
        Q := ∅
        Q.добавить( s )
        Цветs := серый
        d_s := 0
        Пока Q не пуста:
                u := Q.извлечь()
                Для каждой вершины v смежной с u:
                        Если Цвет_v = белый:
                                Цвет_v := серый
                                d_v := d_u + 1
                                Пред_v : =u
                                Q.добавить( v )
                Цвет_u := черный
```

В представленном алгоритме используется очередь вершин *Q*, поддерживающая операции добавления вершины в конец и извлечения из головы очереди. Изначально в очередь заносится исходная вершина *s*, с которой начинается обход. Эта вершина будет извлечена из очереди на первой же итерации цикла, и будут обработаны и помещены в очередь все смежные с ней вершины. Далее из очереди будут извлекаться и аналогичным образом обрабатываться смежные с начальной вершины, затем – смежные им вершины и т.д. Таким образом, в начале очереди будут находиться более близкие к исходной вершине элементы, а в конце – наиболее удаленные.

Поиск в ширину: пример 1/4

Поиск_в_ширину(s):

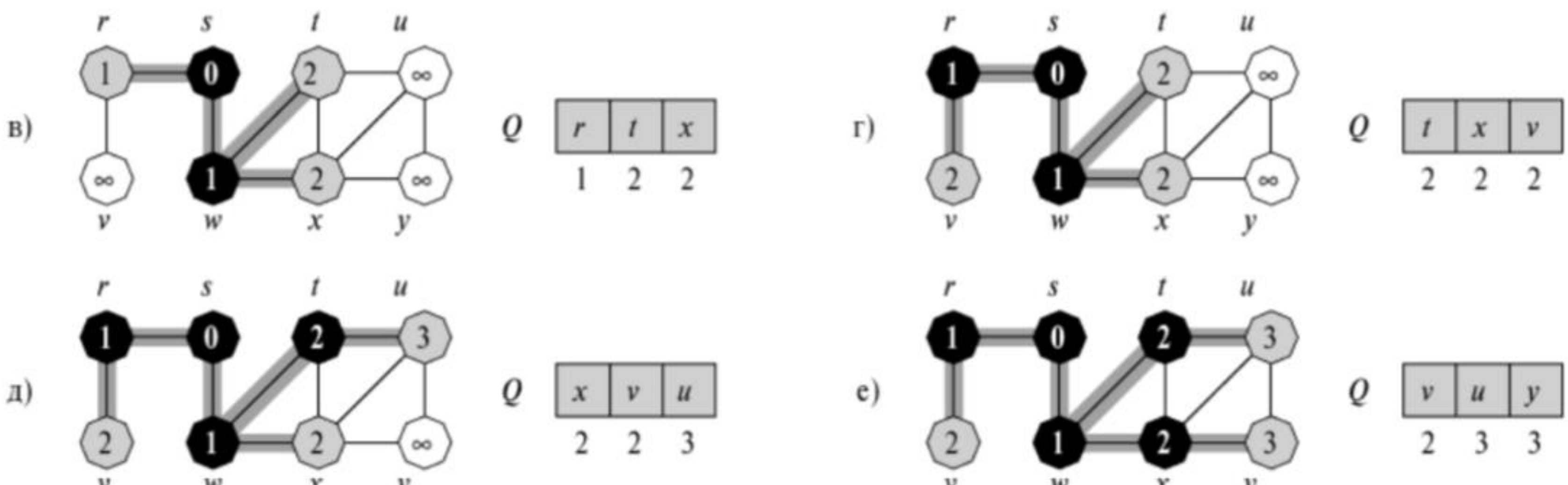
> **Q := ∅**
> **Q.добавить(s)**
> **Цвет$_s$:= серый**
> **d_s := 0**
> **Пока Q не пуста:**
> > **u := Q.извлечь()**
> > **Для каждой вершины v смежной с u:**
> > > **Если Цвет$_v$ = белый:**
> > > > **Цвет$_v$:= серый**
> > > > **d_v := d_u + 1**
> > > > **Пред$_v$: =u**
> > > > **Q.добавить(v)**
> >
> > **Цвет$_u$:= черный**



Поиск в ширину: пример 2/4

Поиск_в_ширину(s):

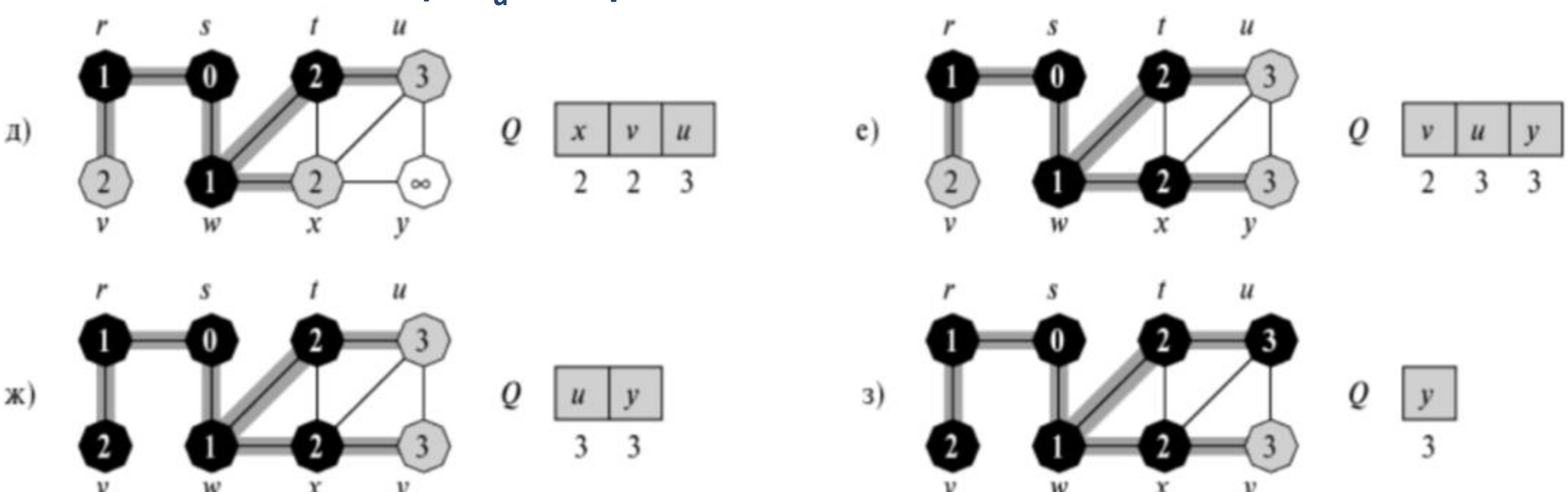
- **Q := ∅**
- **Q.добавить(s)**
- **Цвет$_s$:= серый**
- **d$_s$:= 0**
- **Пока Q не пуста:**
  - **u := Q.извлечь()**
  - **Для каждой вершины v смежной с u:**
    - **Если Цвет$_v$ = белый:**
      - **Цвет$_v$:= серый**
      - **d$_v$:= d$_u$ + 1**
      - **Пред$_v$: =u**
      - **Q.добавить(v)**
  - **Цвет$_u$:= черный**



Поиск в ширину: пример 3/4

Поиск_в_ширину(s):

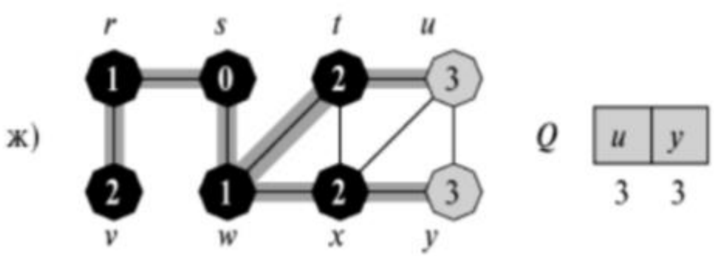
- **Q := ∅**
- **Q.добавить(s)**
- **Цветs := серый**
- **d_s := 0**
- **Пока Q не пуста:**
  - **u := Q.извлечь()**
  - **Для каждой вершины v смежной с u:**
    - **Если Цвет$_v$ = белый:**
      - **Цвет$_v$:= серый**
      - **d_v := d_u + 1**
      - **Пред$_v$: =u**
      - **Q.добавить(v)**
  - **Цвет$_u$:= черный**

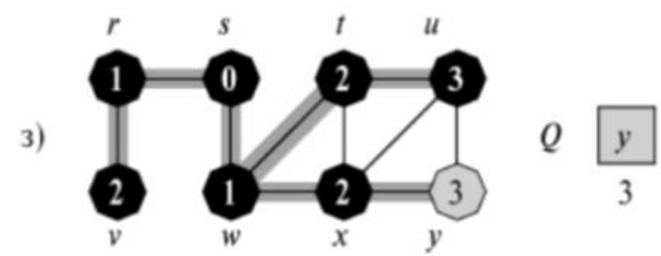


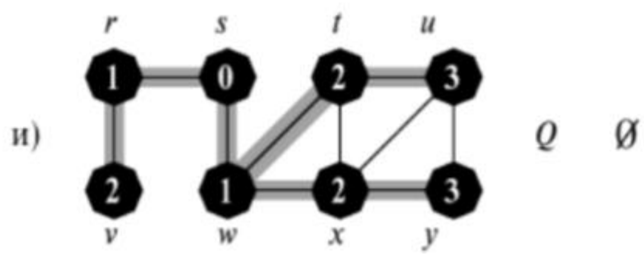
Поиск в ширину: пример 4/4

Поиск_в_ширину(s):

- **Q := ∅**
- **Q.добавить(s)**
- **Цвет$_s$:= серый**
- **d$_s$:= 0**
- **Пока Q не пуста:**
  - **u := Q.извлечь()**
  - **Для каждой вершины v смежной с u:**
    - **Если Цвет$_v$ = белый:**
      - **Цвет$_v$:= серый**
      - **d$_v$:= d$_u$ + 1**
      - **Пред$_v$: =u**
      - **Q.добавить(v)**
  - **Цвет$_u$:= черный**







Задания

Реализовать класс «Граф» для хранения ориентированного графа, в котором каждая вершина имеет уникальный номер (целое число).

В классе реализовать:

1) конструктор по умолчанию,
2) деструктор,

функции

1) обхода в глубину
2) добавления вершины графа
3) удаления вершины графа
4) добавления ребра графа
5) удаления ребра графа

Представление графа организовать в виде списков смежности.