списки (окончание). Графы

Лекция 9, 10

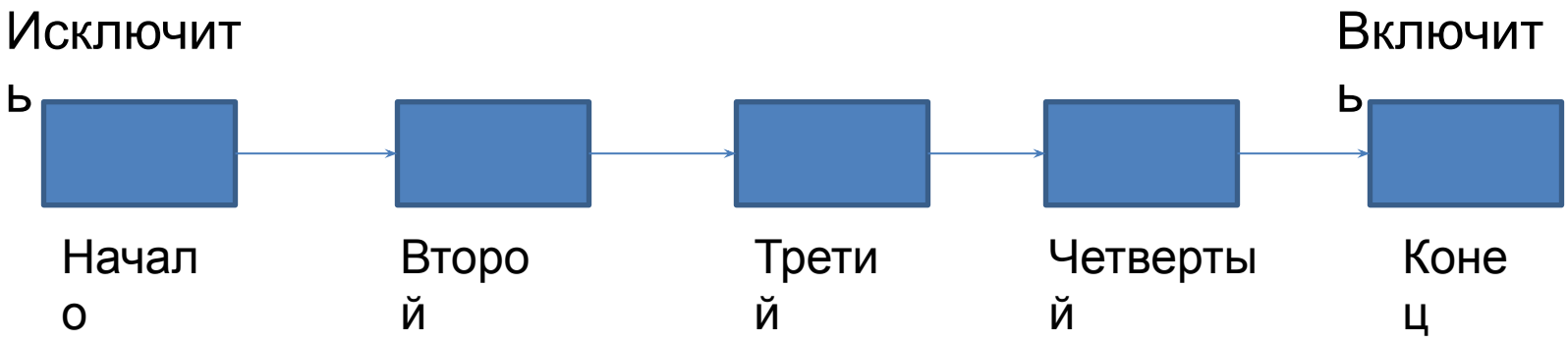
План лекции

- Очередь
  - Реализация с помощью списка
  - Реализация с помощью циклического буфера
- Графы
  - Определения
  - Вычисление кратчайших расстояний с помощью очереди

Очередь

Очередью называется линейный список, в котором все включения производятся на одном конце списка, все исключения – на другом его конце

FIFO (first-in-first-out) – первый вошел, первый вышел



Операции работы с очередью

- create(&Q) – создает очередь
- makeempty (&Q) – делает очередь Q пустой
- front (&Q) – выдает значение первого элемента очереди, не удаляя его
- get(&Q) – выдает значение первого элемента очереди и удаляет его из очереди
- put(&Q, x) – помещает в конец очереди Q новый элемент со значением x
- empty (&Q) -- если очередь пуста, то 1 (истина), иначе 0 (ложь)
- destroy(&Q) -- уничтожает очередь

Реализация очереди с помощью списка

```
struct Element {
    T               value;
    struct Element *    next;
};

struct Queue {
    struct Element *    front;
    struct Element *    back;
};

typedef struct Queue      Queue;
typedef Element *         ptrElement;
```

Create, put

```
void create(Queue *q)
{
    q->front = q->back = NULL;
}
```

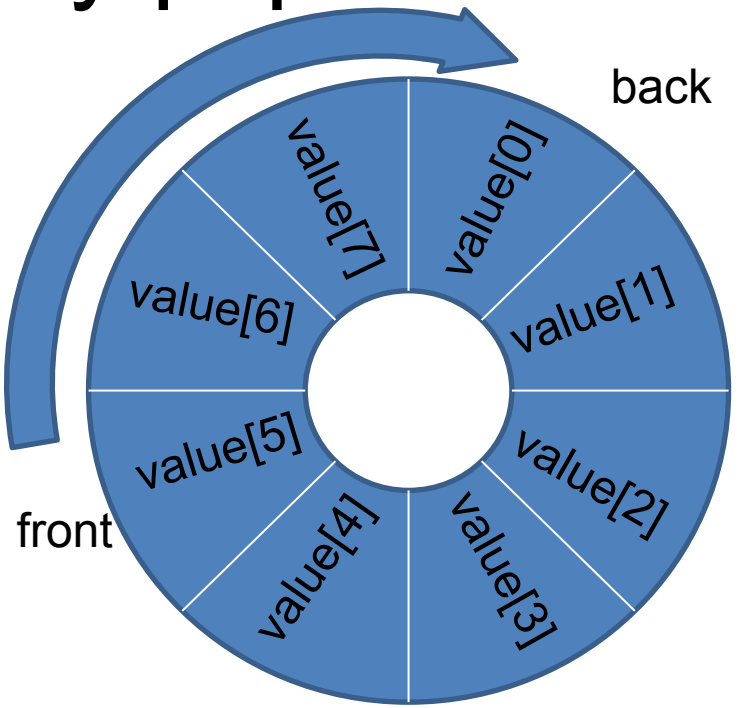
```
void put(Queue *q, T a)
{
    ptrElement p = malloc(sizeof(*p));
    p->value = a;
    p->next = NULL;
    if (q->front == NULL)
        q->front = p;
    else
        q->back->next = p;
    q->back = p;
}
```

Get, empty

```
T get(Queue *q)
{
    ptrElement p = q->front;
    T a = p->value;
    q->front = p->next;
    free(p);
    if (q->front == NULL) q->back = NULL;
    return a;
}
int empty(const Queue *q)
{
    return q->front == NULL;
}
```

Реализация очереди с помощью циклического буфера

```
struct Queue {
    T *value;
    int front;
    int back;
    int size;
};

typedef struct Queue    Queue;
typedef T *             ptrElement;
```



Create, put

```
void create(Queue *q, int size)
{
   q->value = malloc(*q->value*size);
   q->front = q->back = 0;
   q->size = size;
}

void put(Queue *q, T a)
{
   q->value[q->back] = a; // Как узнать, что в очереди нет места?
   q->back = (q->back+1) % q->size;
}
```

Get, empty

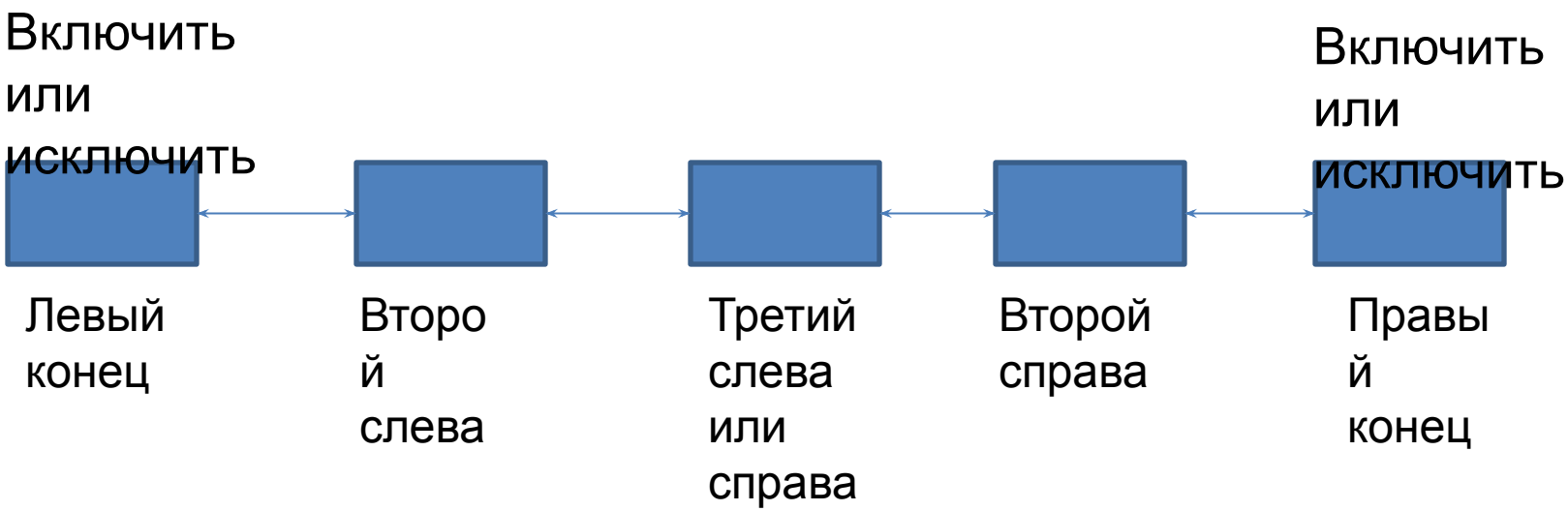
```
T get(Queue *q)
{
    T a = q->value[q->front];
    q->front = (q->front+1) % q->size;
    return a;
}

int empty(const Queue *q)
{
    return q->front == q->back;
}
```

Пример работы с очередью

```
int main()
{
    Queue Q;
    create(&Q, 100);
    put(&Q, 5);
    put(&Q, 7);
    while (!empty(Q))
    {
        int b = get(Q);
        printf("%d\n", b);
    }
    destroy(&Q);
    return 0;
}
```

Дек (double-ended queue) очередь с двумя концами

Деком называется линейный список, в котором включения и исключения производятся на обоих концах списка



Графы

- Очередь
  - Реализация с помощью списка
  - Реализация с помощью циклического буфера
- Графы
  - Определения
  - Вычисление кратчайших расстояний с помощью очереди

Упорядоченная пара

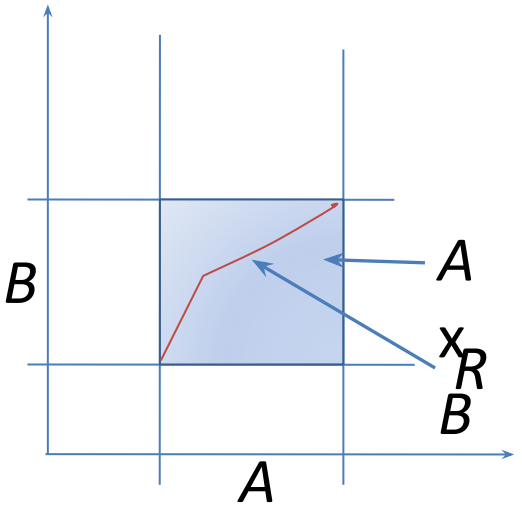
- Пусть A и B – множества
- Упорядоченная пара (a, b), состоящая из $a \in A$ и $b \in B$, это конечное множество {a, {a, b}}
- Упорядоченные пары (a, b) и (c, d) равны, если a = c и b = d
  - Почему?
  - Чем отличается упорядоченная пара от множества {a, b}?

Декартово произведение

- Декартовым произведением AxB множеств A и B называется множество упорядоченных пар { (a, b) | $a \in A$ и $b \in B$ }
- Пример
  A = {1, 2}
  B = {2, 3, 4}
  AxB = {(1, 2), (1, 3), (1, 4), (2, 2), (2, 3), (2, 4)}

Отношения

- Пусть A и B —множества
- Отношением из A в B называется любое подмножество множества AxB
- A называется областью определения отношения R
- B называется множеством значений отношения R
- Факт $(a, b) \in R$ сокращенно записывается aRb
- Отношение $\{(b, a) \mid (a, b) \in R\}$ называется обратным к отношению R и часто обозначается через R^{-1}.



Виды отношений

Пусть A—множество

Отношение R называется на A

- рефлексивным, если aRa для всех a из A
- симметричным, если aRb влечет bRa для a и b из A
- транзитивным, если для любых a, b и c из A из aRb и bRc следует aRc

- Рефлексивное, симметричное и транзитивное отношение называется отношением эквивалентности

- Отношение эквивалентности на множестве A разбивает множество A на непересекающиеся подмножества, называемые классами эквивалентности

- Приведите примеры каждого вида отношений

Графы

- Графом называется пара (A, R), где A — конечное множество, а R — отношение на множестве A
- Элементы A называются вершинами (узлами)
- Элементы R называются дугами (ребрами)
- Если отношение R несимметричное, то граф ориентированный
- Если отношение R симметричное, то граф неориентированный

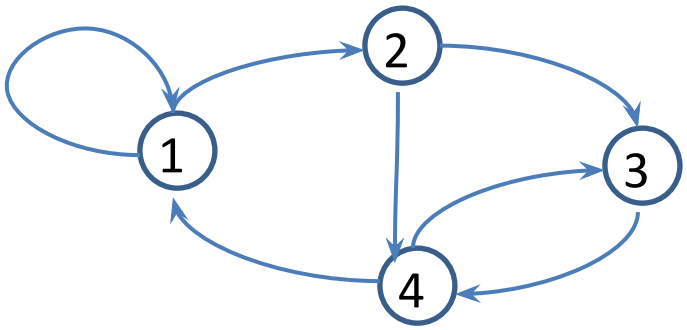
Изображение графов на плоскости

Вершины графа изображают точками

Дуги графа изображают прямо- или криволинейных отрезков

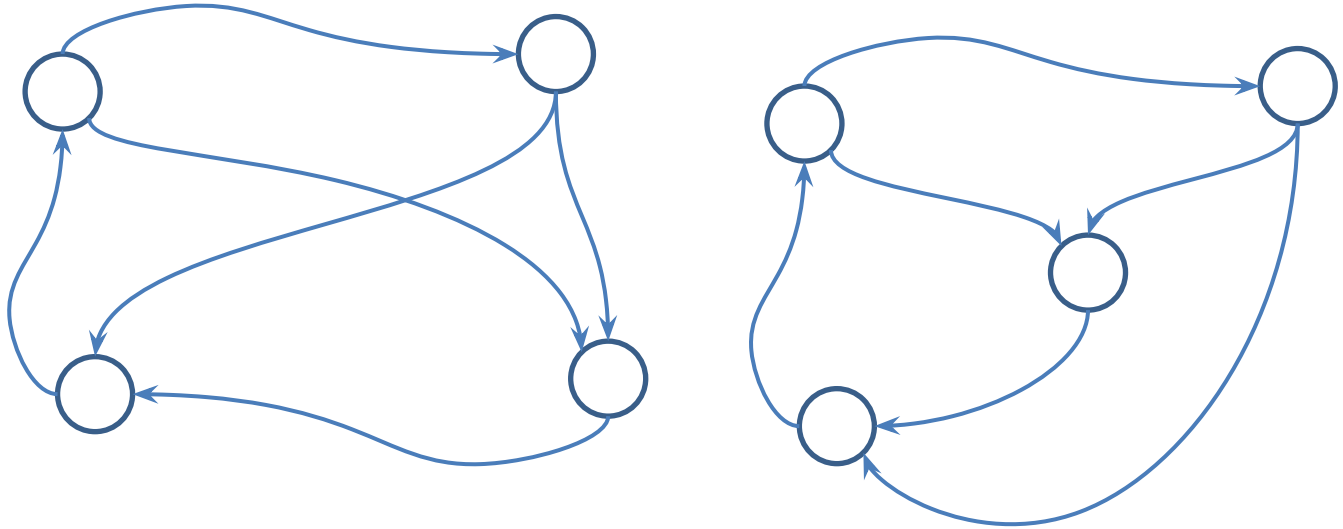
Пример изображения графа на плоскости

A={1, 2, 3, 4} , R = {(1, 1), (1, 2), (2, 3), (2, 4), (3, 4), (4, 1), (4, 3)}



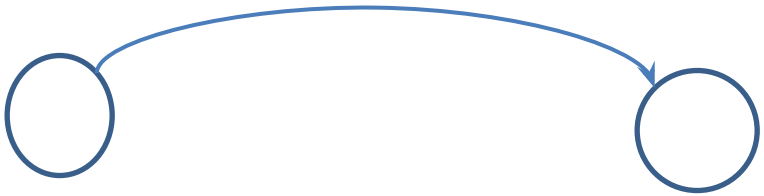
Изображение графов на плоскости

- Изображения дуг графа могут пересекаться -- точки пересечения не являются вершинами
- Граф, который можно изобразить на плоскости без пересечений дуг, называется планарным.
- Пример различных изображений графа на плоскости
  A={1, 2, 3, 4} , R = {(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (4, 1)}



Дуги графа

- Пара $(a, b) \in R$ называется дугой (ребром) графа G
- Дуга выходит из вершины a и входит в вершину b
- Вершина a предшествует вершине b, а вершина b следует за вершиной a
- Вершина b смежна с вершиной a



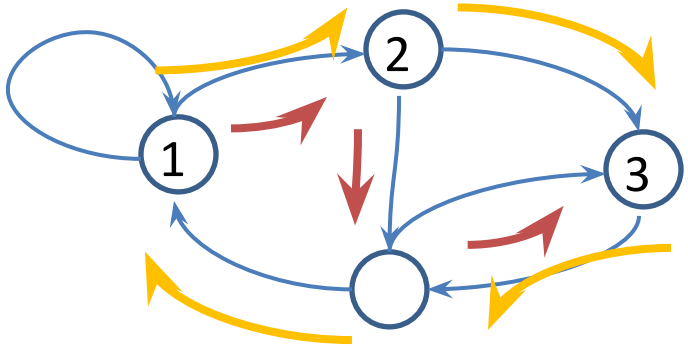
Путь и цикл в графе

- Последовательность вершин (a_0, a_1, ... , a_n), $n \geq 1$, называется путем (маршрутом) длины n из вершины a_0 в вершину a_n, если для каждого $1 \leq i \leq n$ существует дуга, выходящая из вершины a_{i-1} и входящая в вершину a_i
- Если существует путь из вершины a_0 в вершину a_n, то говорят, что a_n достижима из a_0
- Циклом называется путь (a_0, a_1, ...,a_n), т.ч. $a_0 = a_n$

Путь и цикл в графе

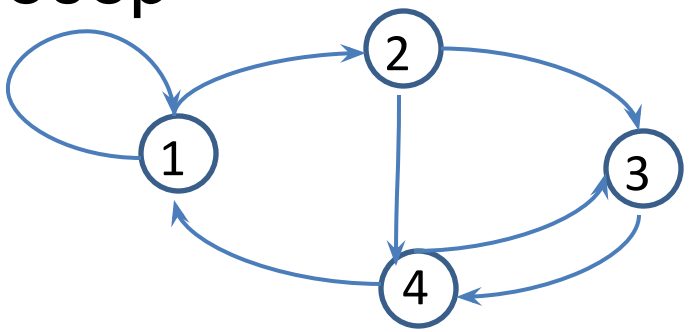
- A={1, 2, 3, 4}
- R = {(1, 1), (1, 2), (2, 3), (2, 4), (3, 4), (4, 1), (4, 3)}

- Путь (1, 2, 4, 3)
- Цикл (1, 2, 3, 4, 1)



Степень вершины

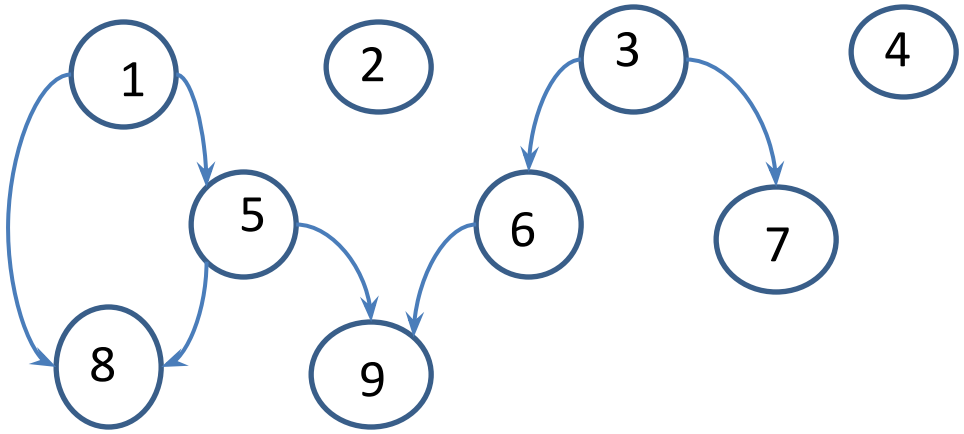
- Степенью по входу (полустепенью входа) вершины называется число входящих в нее дуг
- Степенью по выходу (полустепенью исхода) вершины называется число выходящих из нее дуг
- Если граф неориентированный, то степенью вершины называется количество инцидентных с ней ребер



Для вершины 2:
полустепень входа = 1
полустепень исхода = 2

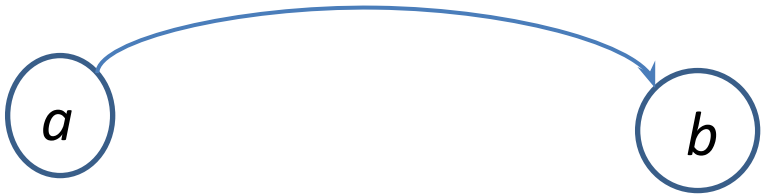
Ациклические графы

- Ациклическим графом называется (ориентированный) граф, не имеющий циклов
- Вершина, степень по входу которой равна 0, называется базовой
- Вершина, степень по выходу которой равна 0, называется листом (концевой вершиной)



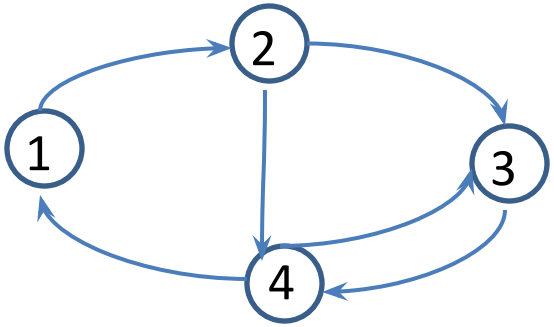
Дуга и путь в ациклическом графе

- Пусть (a, b) – дуга в ациклическом графе
- Вершина a называется прямым предком b, b называется прямым потомком a
- Если существует путь из a в b, то a называется предком b, b называется потомком a



Матрица смежностей

- Пусть дан граф G = (V,E), N = |V|, M = |E|
- Матрица смежностей для графа G – это матрица A размера NxN, состоящая из 0 и 1, в которой A[i, j]=1 тогда и только тогда, когда есть ребро из узла i в узел j

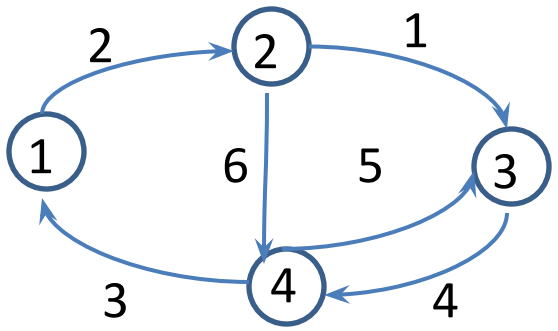


	1	2	3	4
1	0	1	0	0
2	0	0	1	1
3	0	0	0	1
4	1	0	1	0

Матрица инцидентностей

Матрица инцидентностей для графа G – это матрица B размера NxM, в которой

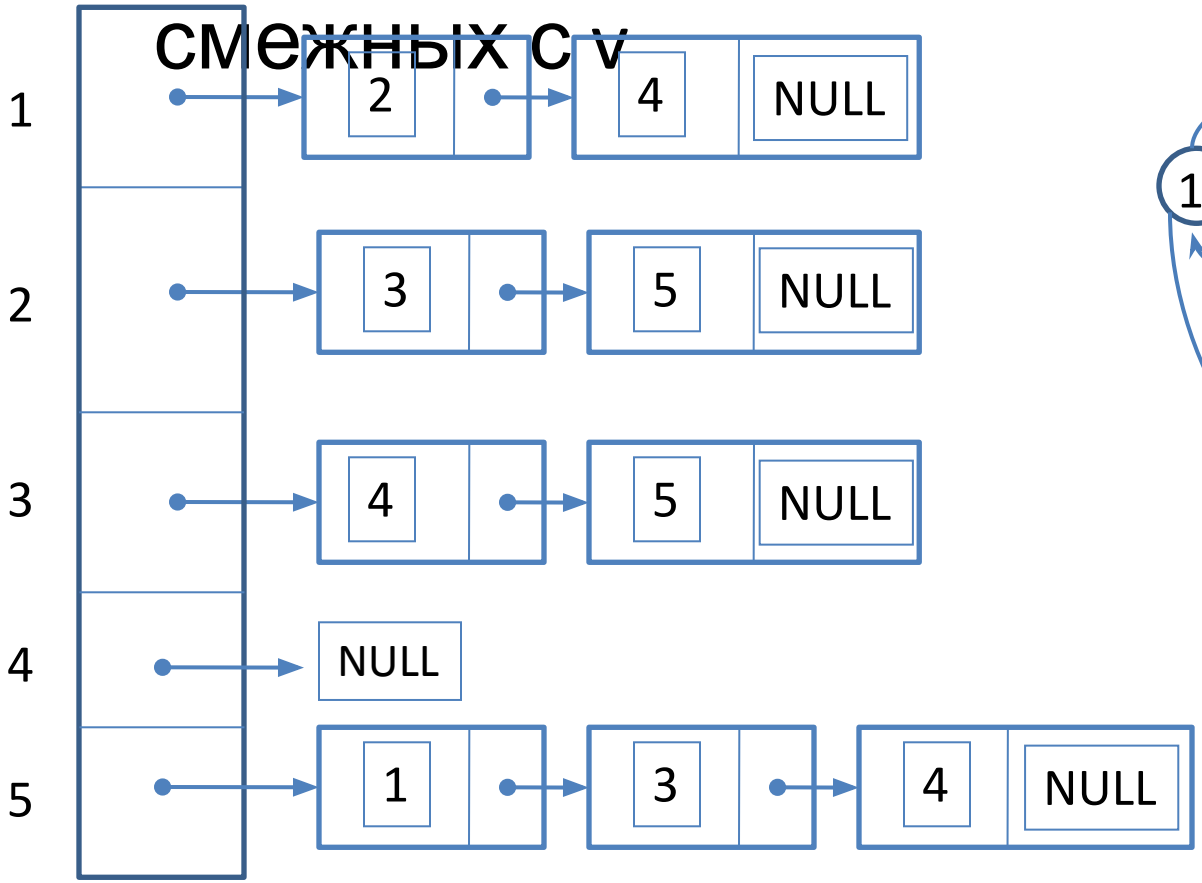
$$B[i, j] = \begin{cases} 1, & \text{если ребро j выходит из вершины i} \\ -1, & \text{если ребро j входит в вершину i} \\ 0, & \text{если ребро j не связано с вершиной i} \end{cases}$$

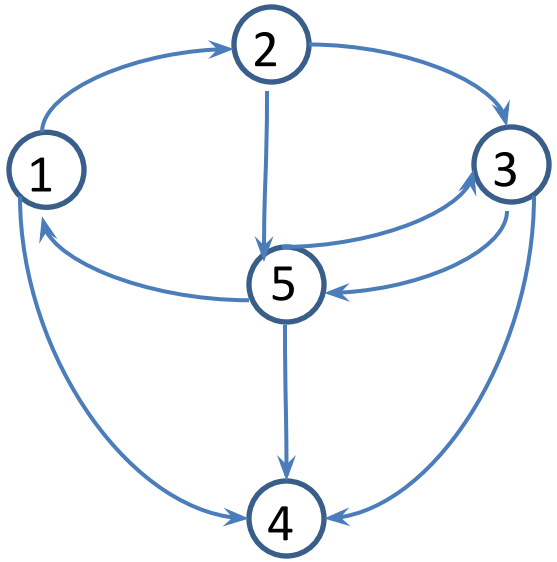


	1	2	3	4	5	6
1	0	1	-1	0	0	0
2	1	-1	0	0	0	1
3	-1	0	0	1	-1	0
4	0	0	1	-1	1	-1

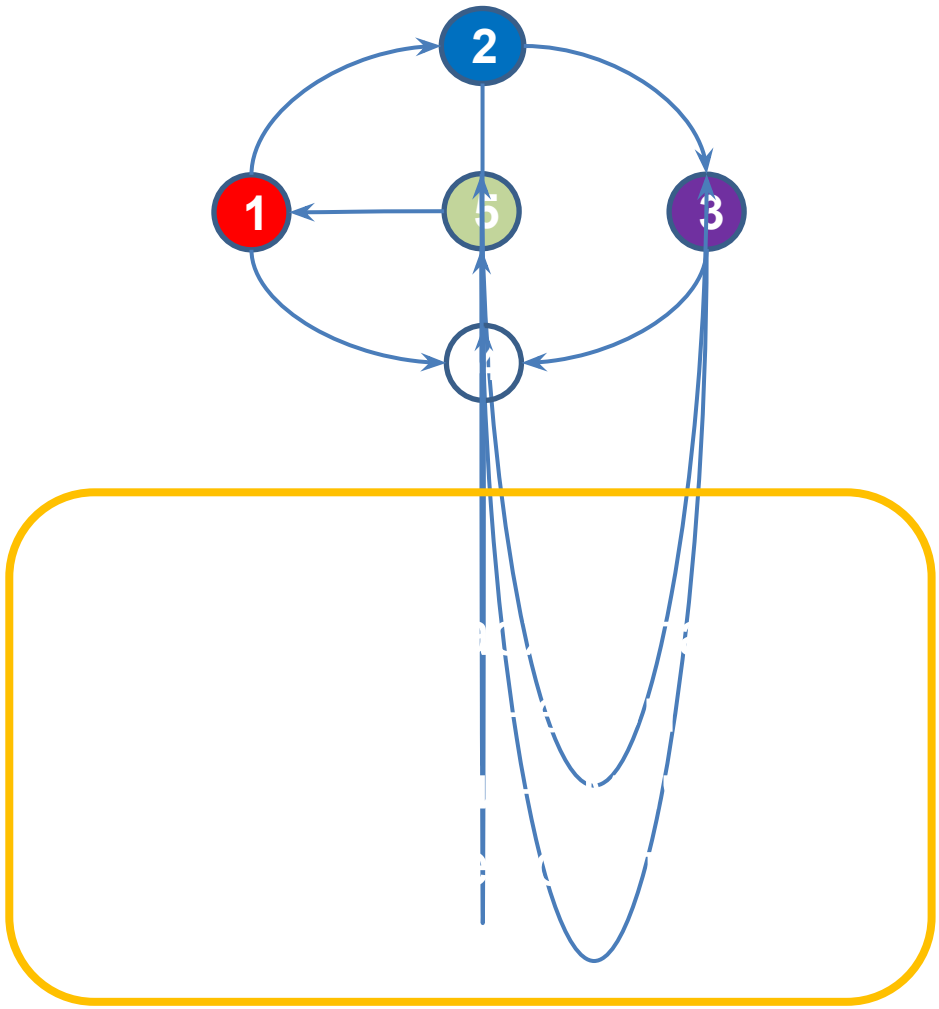
Списки смежностей

Списком смежностей для узла v называется список всех узлов w, смежных с v





Табличное представление списков смежностей



	Номер вершины	Следующий
1		6
2		8
3		10
4		0
5		12
6	2	7
7	4	0
8	3	9
9	5	0
10	4	11
11	5	0
12	1	13
13	3	14
14	4	0

Поиск в ширину в графе

- Способ нумерации вершин произвольного графа (один из)
- Проектирование ИС и печатных плат, ...
- Основа большого числа алгоритмов
  - Поиск кратчайших путей
  - Вычисление максимального потока в графе
  - Проверка связности
- Breadth-first search, BFS

Алгоритм поиска в ширину

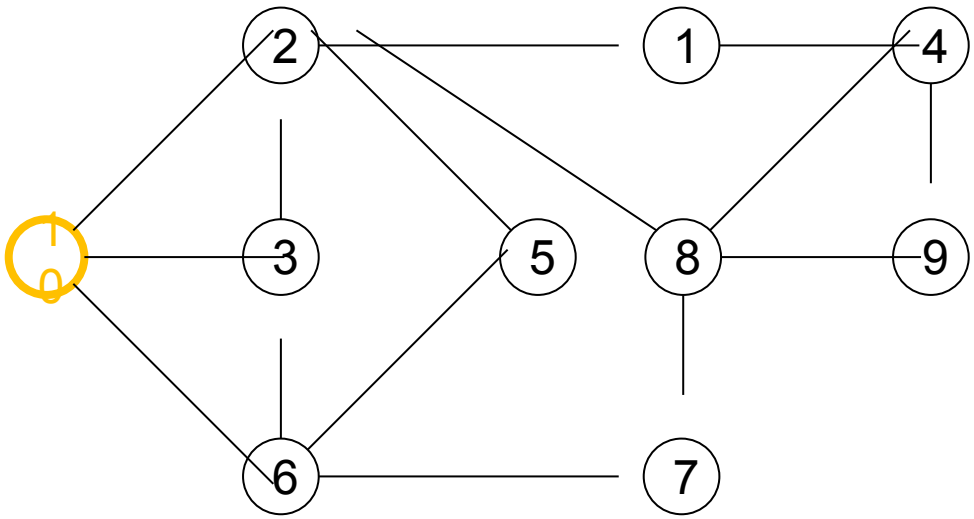
- Пусть дан граф G и выбрана некоторая его вершина s
- Поиск в ширину вычисляет для каждой вершины u два номера
  - [illegible]
  - d[u] кратчайшее расстояние от s до u

- Схема алгоритма
  - Шаг 1: d[s] = 0
  - Шаг n: обрабатываем все вершины на расстоянии n-1 от s
    - Каждого соседа v вершины u с пометкой d[u] = *n*-1 нумеруем [illegible] и d[v] = n

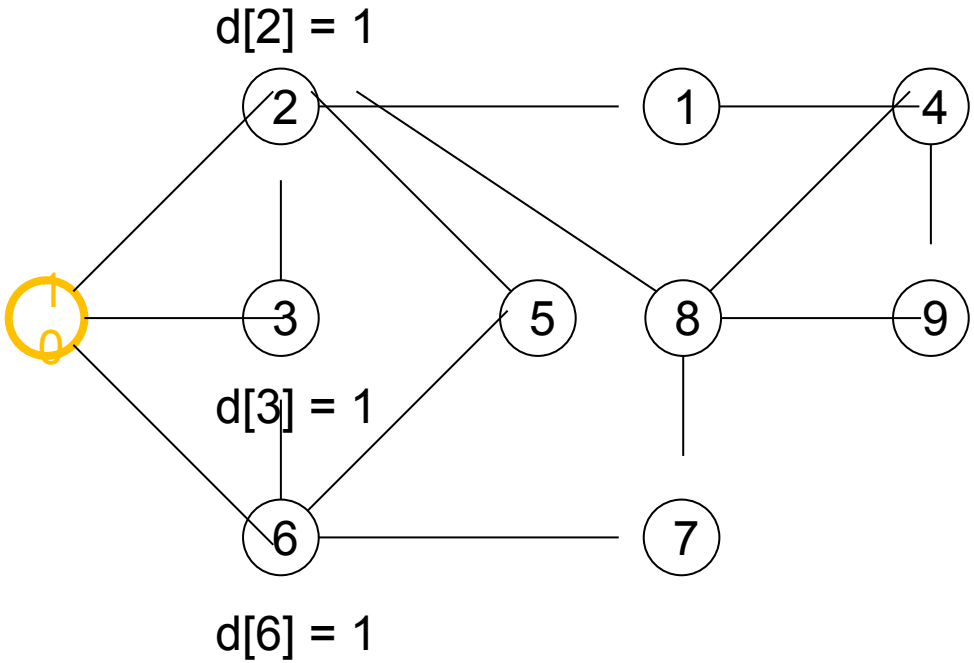
Алгоритм поиска в ширину

```
BFS (матрица смежности граф G, число вершин n, вершина s)
{
    for (u = 0; u < n; u++)
        d[u] = n; // почему?
    d[s] = 0;
    put(s, Q);
    while (! empty(Q)) {
        u = get(Q);
        for (v = 0; v < n; v++) if (G[v][u] == 1) { // сосед u
            if (d[v] > d[u]+1) {
                d[v]= d[u]+1;
                put(Q, v);
        }}
    }
}
```

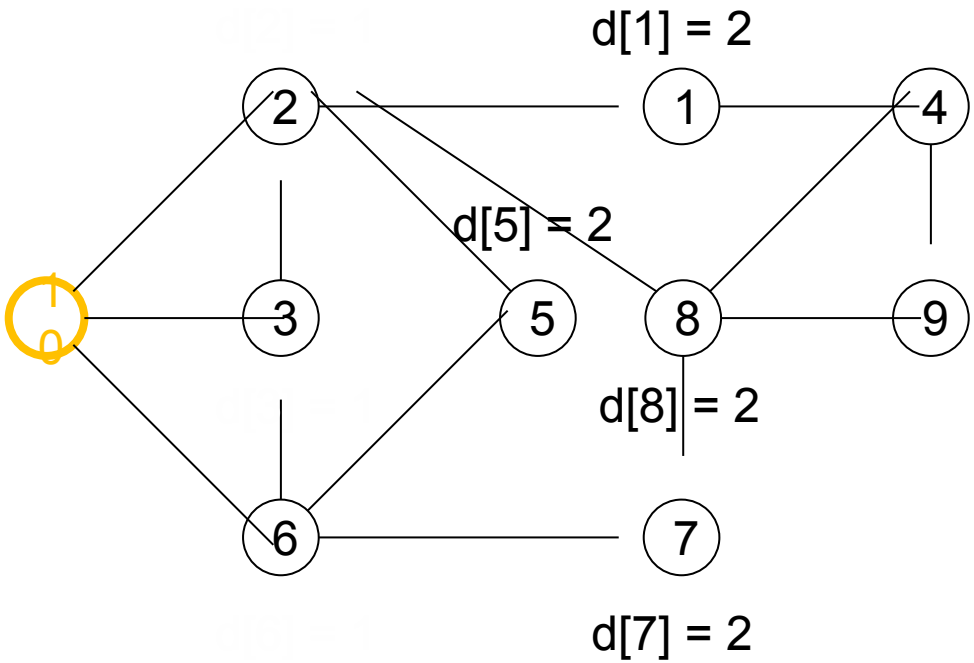
Метод поиска в ширину



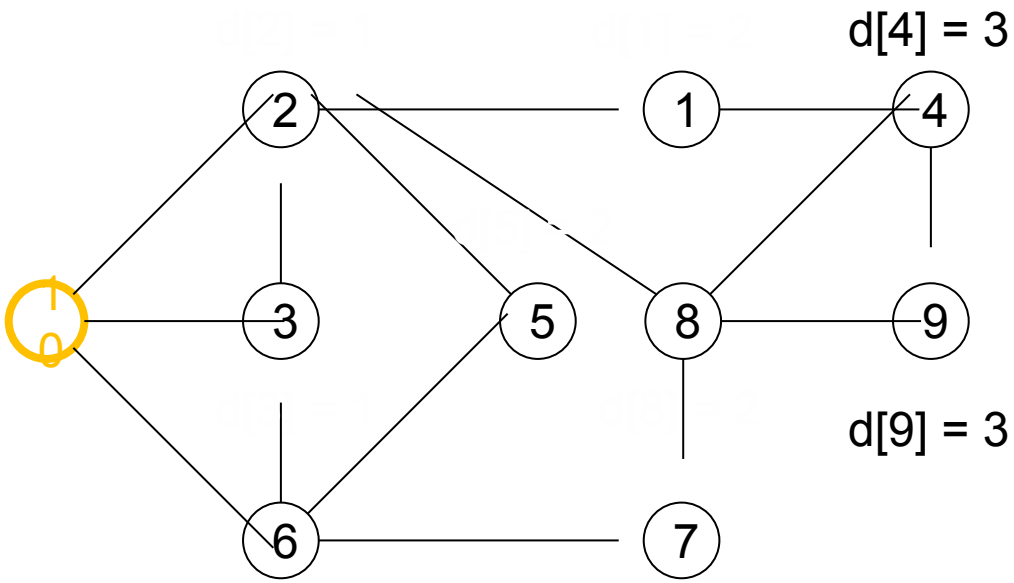
Метод поиска в ширину



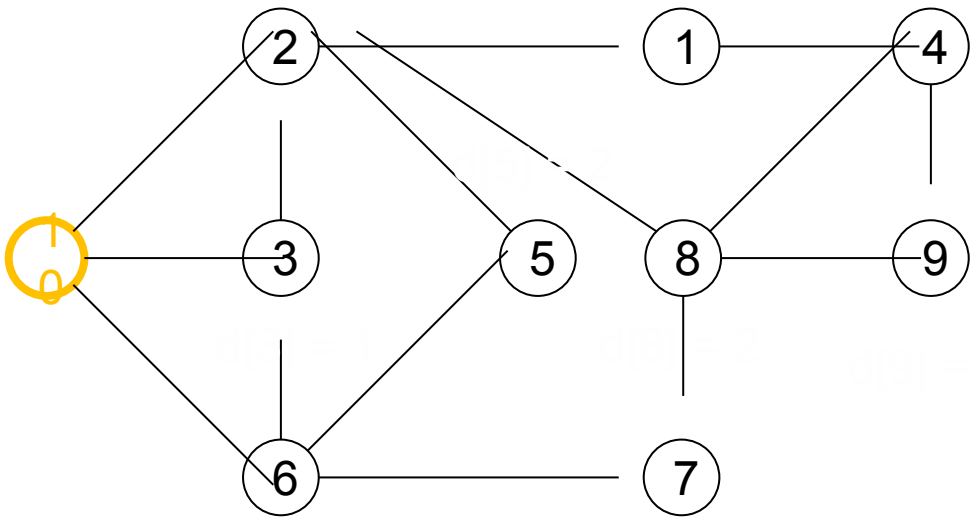
Метод поиска в ширину



Метод поиска в ширину



Метод поиска в ширину



V	Расстояние до 10	Путь через
1	2	2
2	1	
3	1	
4	3	2, 1
5	2	2 или 6
6	1	3
7	2	6
8	2	2
9	3	2, 8

Заключение

- Очередь
  - Реализация с помощью списка
  - Реализация с помощью циклического буфера
- Графы
  - Определения
  - Вычисление кратчайших расстояний с помощью очереди