

Лекция 10

Графы. Обходы графов

Курносов Михаил Георгиевич

E-mail: mkurnosov@gmail.com

WWW: www.mkurnosov.net

Курс «Структуры и алгоритмы обработки данных»

Сибирский государственный университет телекоммуникаций и информатики (Новосибирск)

Весенний семестр, 2016

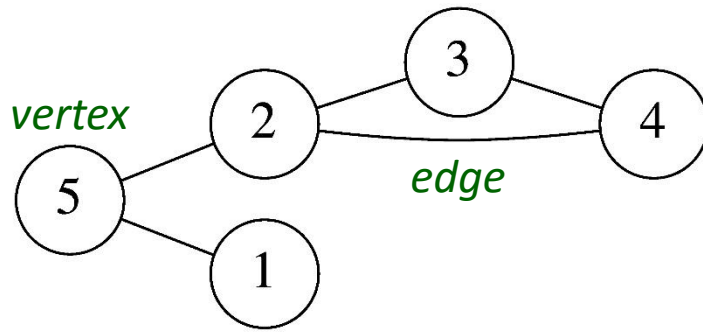
Понятие графа

- **Граф (graph)** – это совокупность непустого множества V вершин и множества E ребер

$$G = (V, E),$$

$$n = |V|, \quad m = |E|,$$

$$V = \{1, 2, \dots, n\}, \quad E = \{(u_1, v_1), (u_2, v_2), \dots, (u_m, v_m)\}$$



$$V = \{1, 2, 3, 4, 5\}$$

$$E = \{(1, 5), (2, 5), (2, 3), (2, 4), (3, 4)\}$$

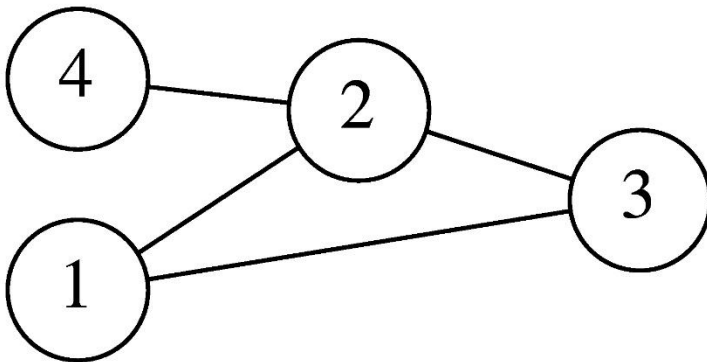
$(1, 5)$ и $(5, 1)$ – это одно и тоже ребро

Виды графов

Графы (graphs)

Неориентированные графы (undirected graphs)

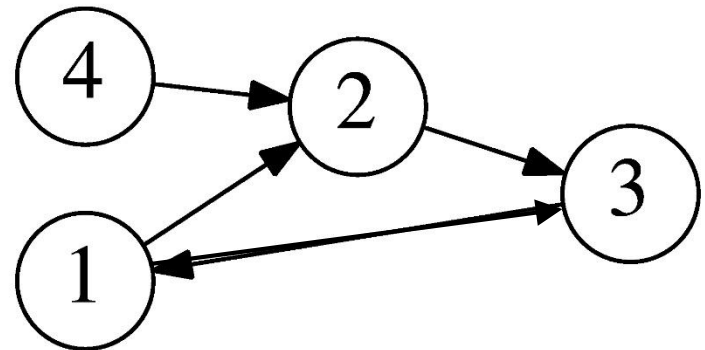
Ребра (edges) не имеют направлений



$(1, 2)$ и $(2, 1)$ – одно и то же ребро

Оrientированные графы (directed graphs)

Ребра – дуги (arcs, edges) имеют направления



$(1, 3)$ и $(3, 1)$ – разные дуги!

Основные определения

Вершина (vertex, node)

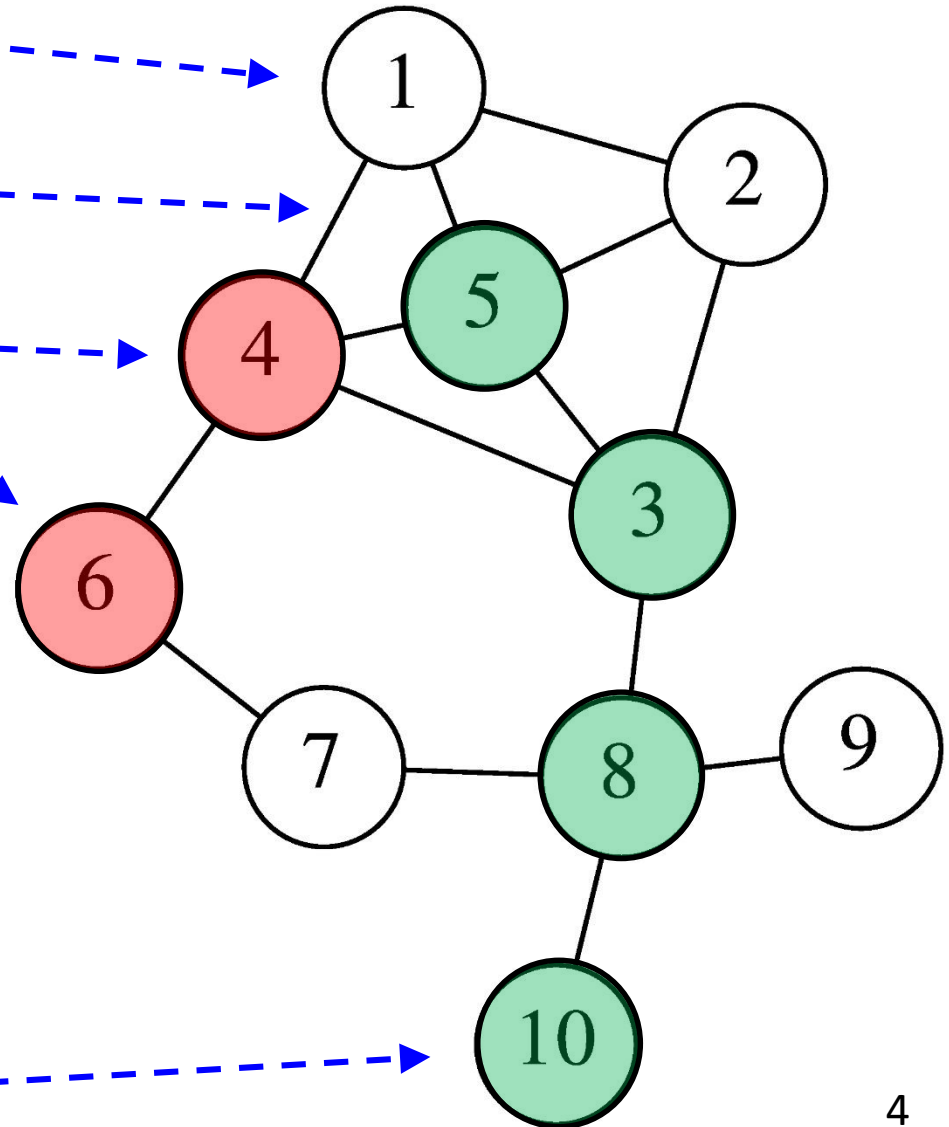
Ребро (edge, link)

Смежные вершины
(adjacent vertices)

Ребро (4, 6) *инцидентно*
(incident) вершинам 4 и 6

Путь (path) – последовательность
вершин, в которой следующая
вершина (после первой) является
смежной с предыдущей
(все вершины и ребра в пути
различны)

Путь: (10, 8, 3, 5)



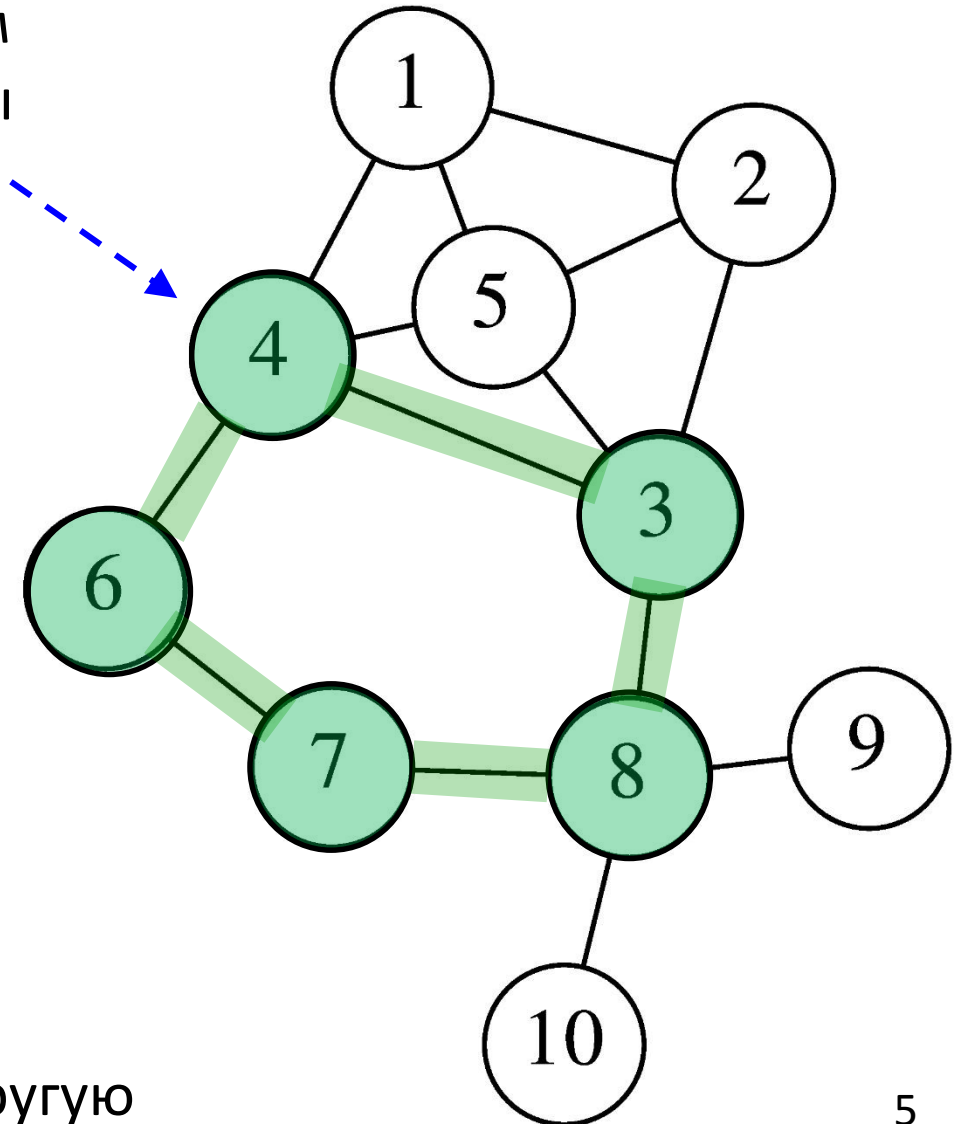
Основные определения

Цикл (cycle) – путь, в котором первая и последняя вершины совпадают: (4, 6, 7, 8, 3, 4)

Степень вершины
(vertex degree) –
количество ребер,
инцидентных вершине

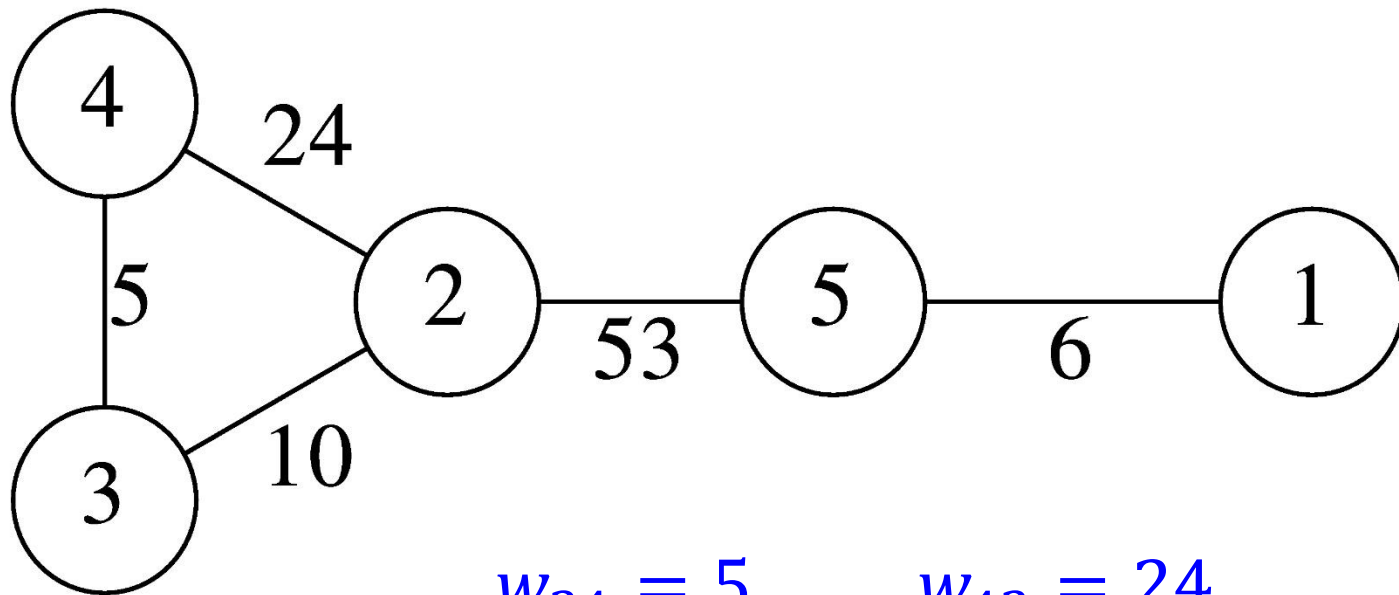
$\deg(7) = 2$, $\deg(1) = 3$

Связный граф
(connected graph) – граф,
в котором существует путь из
каждой вершины в любую другую



Основные определения

- **Взвешенный граф** (weighted graph) – это граф, ребрами (дугам) которого назначены веса
- Вес ребра (i, j) обозначим как w_{ij}



$$w_{34} = 5, \quad w_{42} = 24, \quad \dots$$

Основные определения

Полный граф (complete graph) – это граф, в котором каждая пара различных вершин смежна (каждая вершина соединена со всеми)

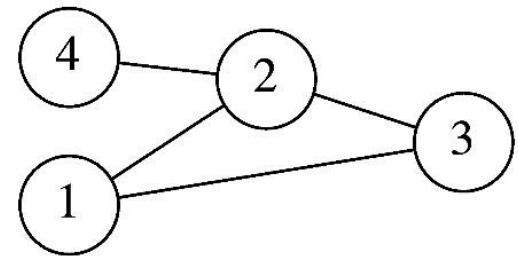
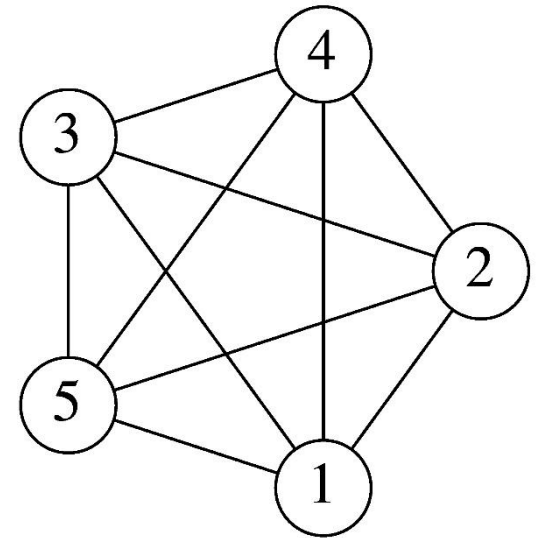
Количество ребер в полном неориентированном графе:

$$m = \frac{n(n - 1)}{2}$$

Насыщенность D графа (density):

$$D = \frac{2m}{n(n - 1)}$$

У полного графа насыщенность $D = 1$



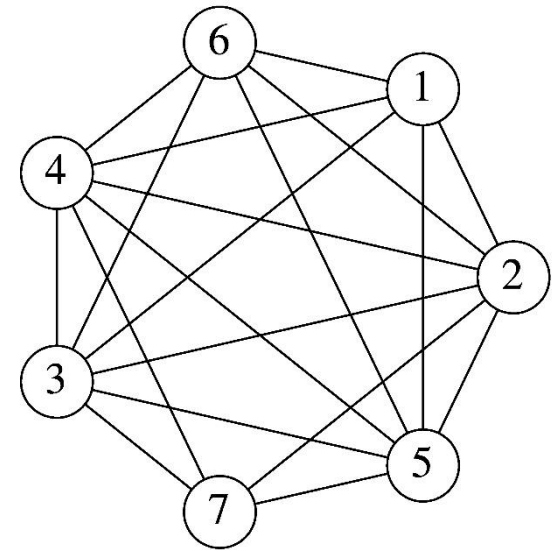
$$D = \frac{\frac{4 \cdot 3}{2}}{4 \cdot 3} = \frac{4}{6} = 0.66$$

Основные определения

Насыщенный граф (dense graph) – это граф, в котором количество ребер близко к максимально возможному

$$|E| = O(|V|^2)$$

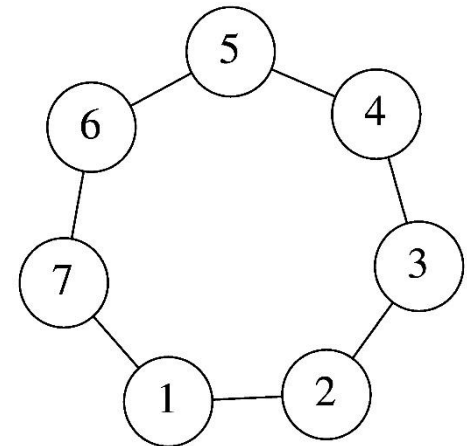
$$D = \frac{2 \cdot 19}{7 \cdot 6} = 0.9, \quad D > 0.5$$



Разреженный граф (sparse graph) – граф, в котором количество ребер близко к количеству вершин в графе

$$|E| = O(|V|)$$

$$D = \frac{2 \cdot 7}{7 \cdot 6} = 0.33, \quad D < 0.5$$



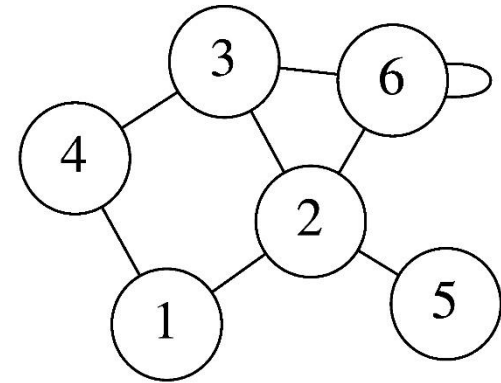
Представление графов в памяти

- Представление графа в памяти (формат его хранения) определяет вычислительную сложность операций над графом и объем требуемой памяти
- Основные способы представления графов в памяти:
 - ❑ **Матрица смежности** (adjacency matrix) – эффективна для насыщенных графов
 - ❑ **Списки смежных вершин** (adjacency list) – эффективен для разреженных графов

Матрица смежности

- **Матрица A смежности**
(adjacency matrix) – это матрица $n \times n$ элементов, в которой

$$a_{ij} = \begin{cases} 1, & \text{если } (i, j) \in E, \\ 0, & \text{иначе.} \end{cases}$$



- Объем требуемой памяти $O(|V|^2)$
- Быстрое определение присутствия ребра (i, j) в графе
- За время $O(1)$ получаем доступ к элементу a_{ij} матрицы

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Матрица смежности

- Какого размера граф можно разместить в оперативной памяти объемом 8 Гб используя матрицу смежности?

```
int a[n][n];
```

Матрица смежности

- Какого размера граф можно разместить в оперативной памяти объемом 8 Гб используя матрицу смежности?

```
int a[n][n];
```

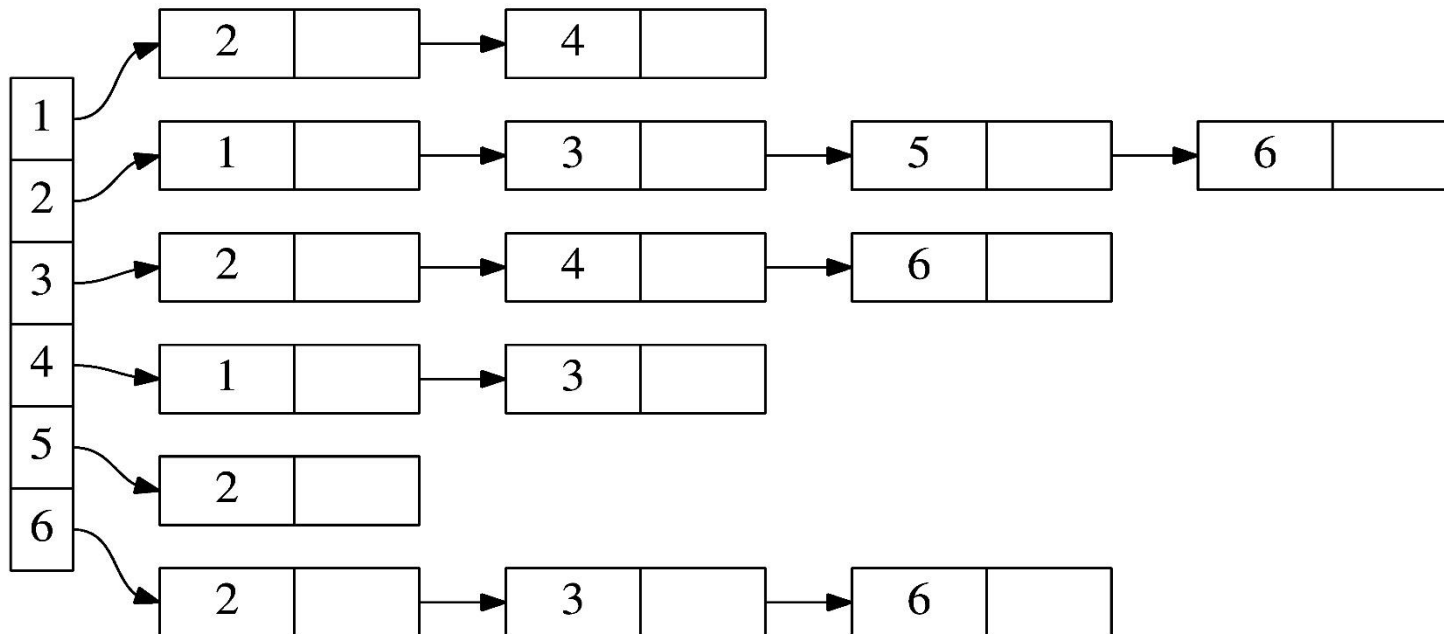
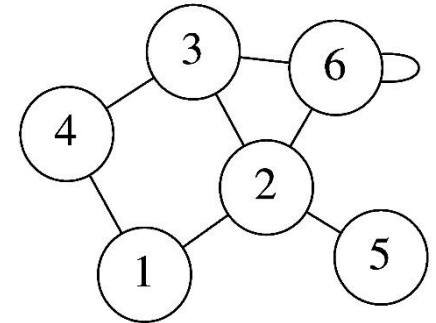
- `sizeof(int) = 4` байта
- $8 \text{ Гб} = 8 \cdot 2^{30}$ байт
- $8 \cdot 2^{30} / 4 = 2 \cdot 2^{30}$ – можно разместить $2^{31} = 2\,147\,483\,648$ элементов типа `int`
- $n = \lfloor \sqrt{2^{31}} \rfloor = 46340$ – количество строки и столбцов

```
int a[46340][46340];
```

- Надо учесть, что часть памяти занята ОС и другими программами (предполагаем, что доступно 90% памяти (~ 7 Гб), тогда $n = 43\,347$)
- Сколько поместится элементов типа `unsigned char` или `uint8_t`?

Списки смежных вершин

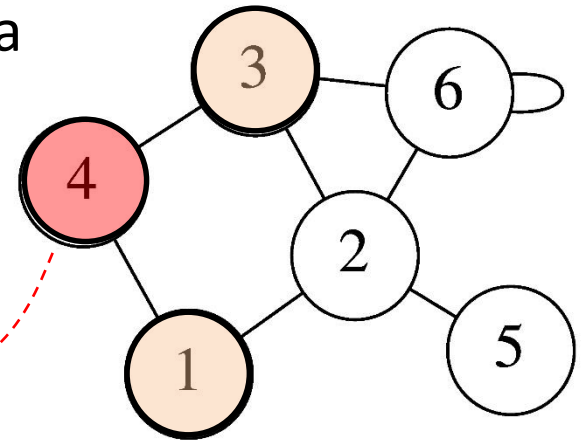
- **Списки смежных вершин** (adjacency list) – это массив $A[n]$, каждый элемент $A[i]$ которого содержит список узлов смежных с вершиной i



Эффективен для разреженных графов ($|E| \approx |V|$)

Списки смежных вершин

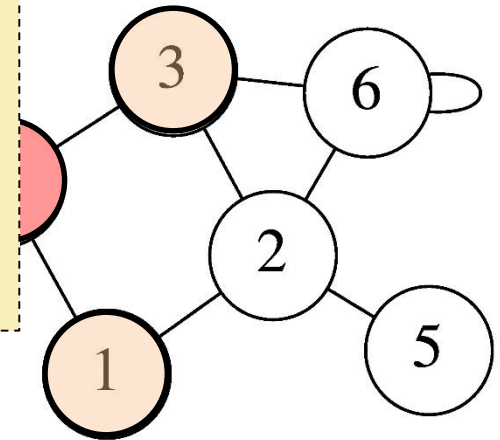
- Реализация списка смежных вершин на основе массивов $A[n + 1]$ и $L[2m]$
- Список смежных вершин узла i :
 $L[A[i]], L[A[i] + 1], \dots, L[A[i + 1] - 1]$



Списки смежных вершин

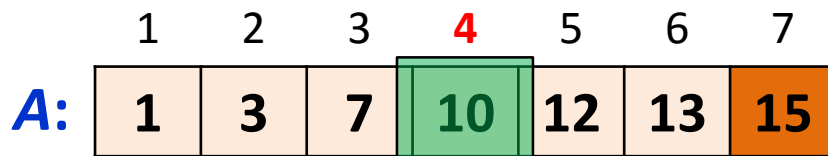
```
// Обход смежных вершин узла i
for (j = A[i]; j < A[i + 1]; j++) {
    v = L[j];
    // Обработать вершину v
}
```

$T = O(m)$

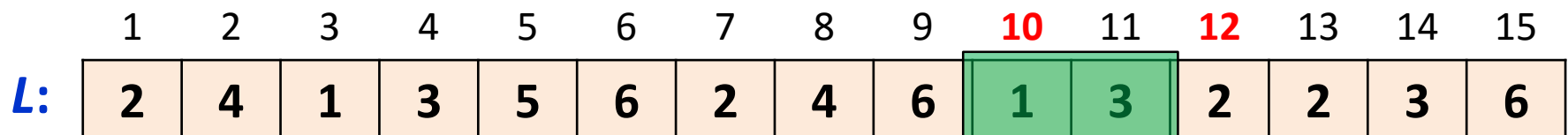


- Количество смежных узлов вершины i :
 $A[i + 1] - A[i]$

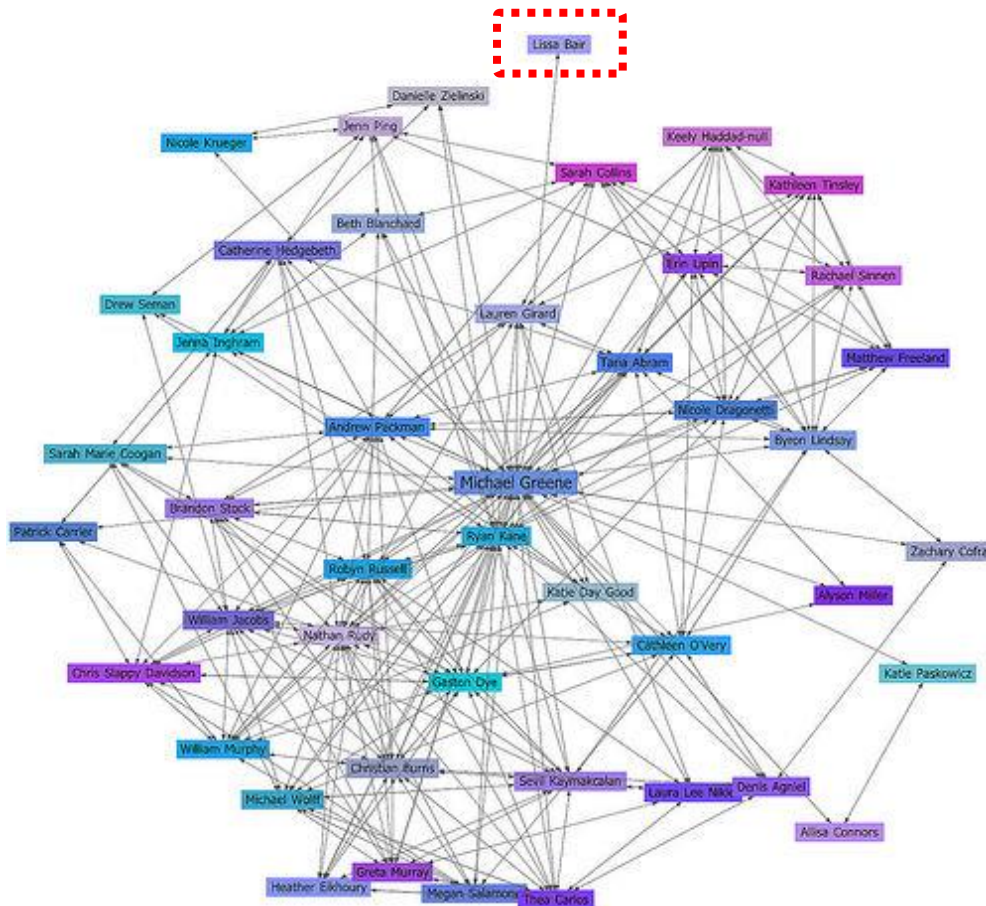
Ошибка из-за петли!



– индексы начала и конца списка смежных вершин в массиве L



Графы в реальной жизни

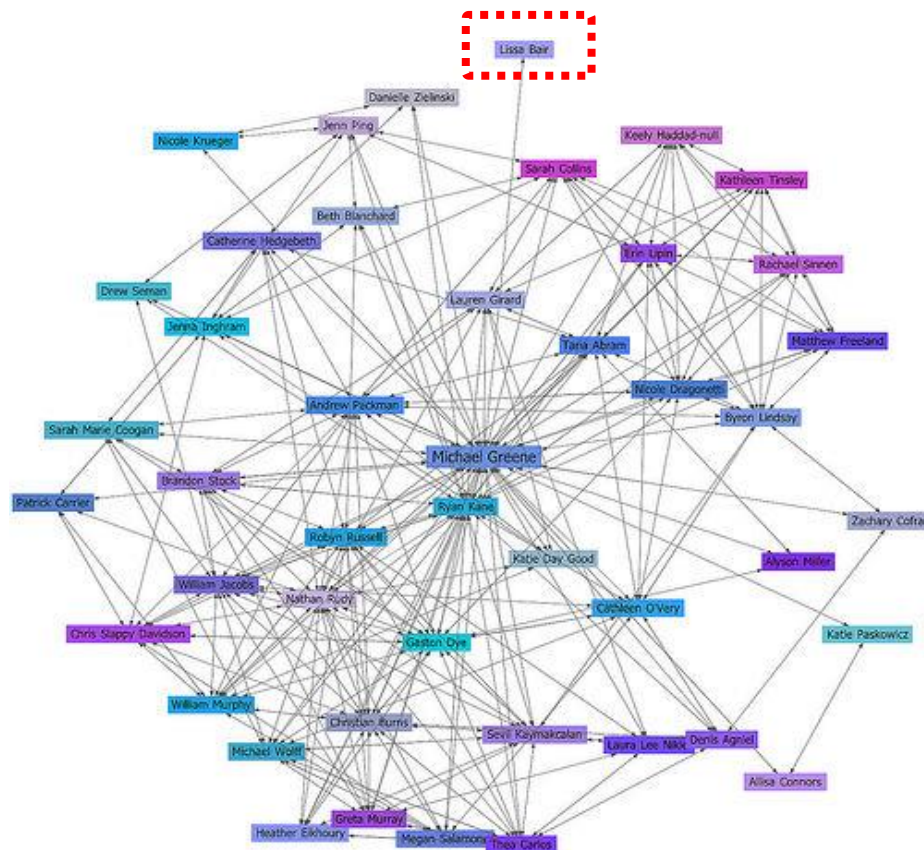


- Задан начальный пользователь (узел графа)
- Как начиная со стартового узла найти пользователя с заданным именем?
- Как перебрать всех пользователей и применить к ним некоторую процедуру?
 - ☐ изменить профиль
 - ☐ вычислить число пользователей старше 20 лет

Граф социальной сети

Графы в реальной жизни

- **Обход графа** (graph traversal) – это процедура перебора (посещения) всех вершин графа начиная с заданной



Граф социальной сети

Поиск в глубину (depth-first search)

- **Поиск в глубину** (depth-first search – DFS) – процедура посещения всех вершин графа начиная с заданного узла v
- Сперва посещаем (обрабатываем) все самые “глубокие” вершины

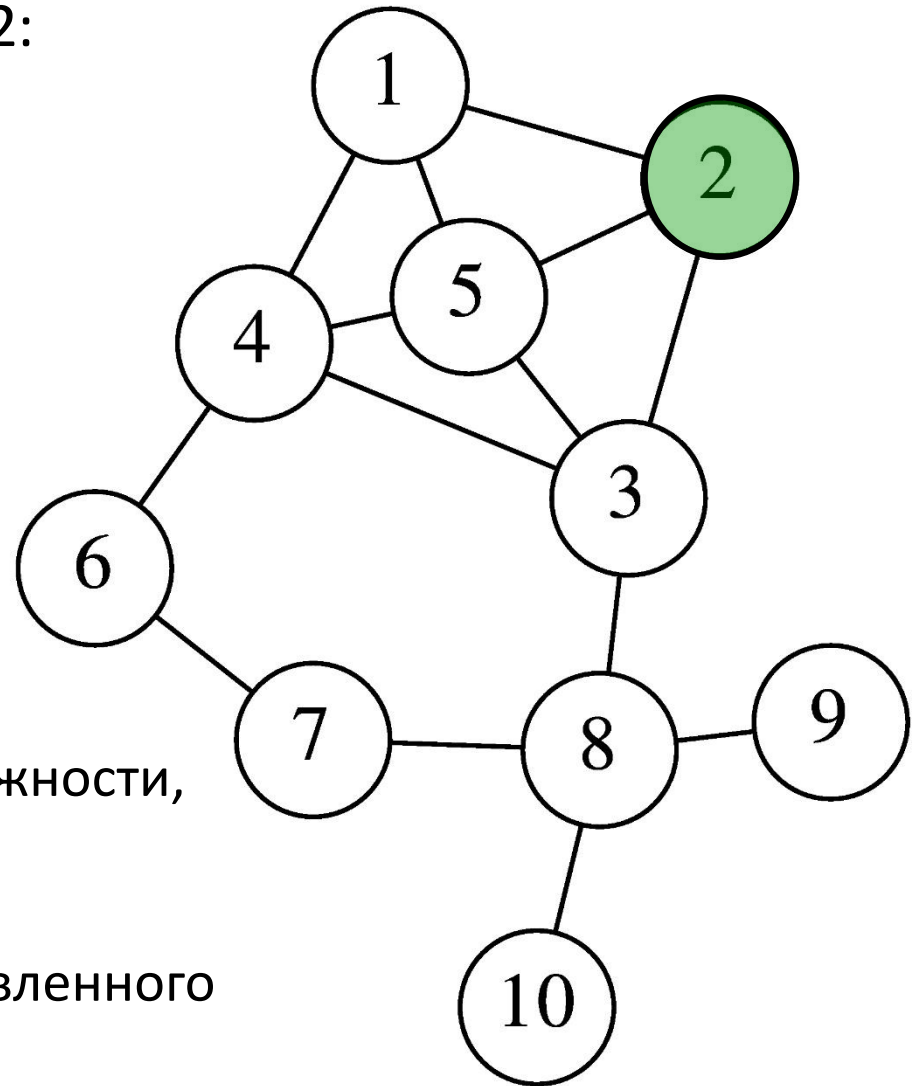
```
function DFS(v)
    visited[v] = true
    // Обрабатываем данные вершины v
    for each u in Adj(v) do    // Перебор смежных вершин
        if visited[u] = false then
            DFS(u)    // Рекурсивно обрабатываем вершину u
        end if
    end for
end function
```

Поиск в глубину (depth-first search)

- Обход в глубину с вершины 2:
DFS(2)

```
for each u in Adj(2) do  
    ...  
end for
```

- Обход в глубину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
- Обход в глубину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$

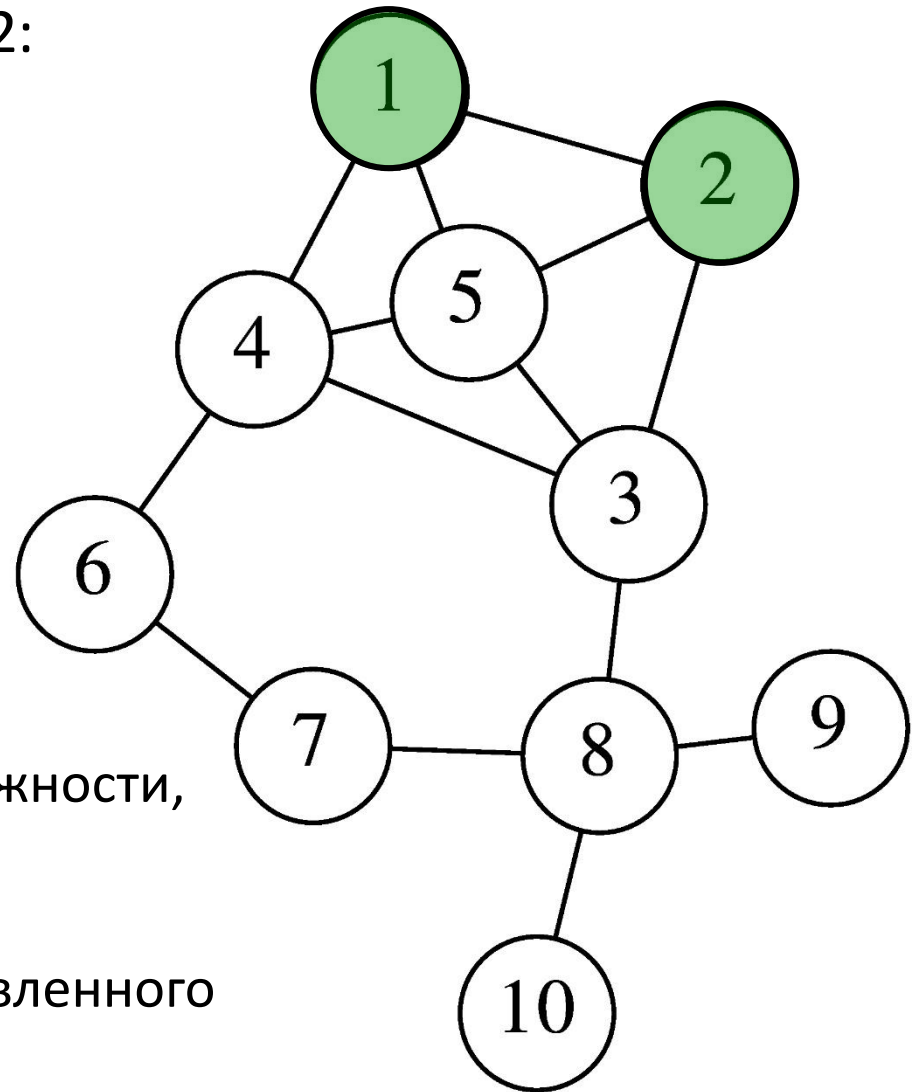


Поиск в глубину (depth-first search)

- Обход в глубину с вершины 2:
DFS(2)

```
for each u in Adj(2) do  
    ...  
end for
```

- Обход в глубину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
- Обход в глубину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$

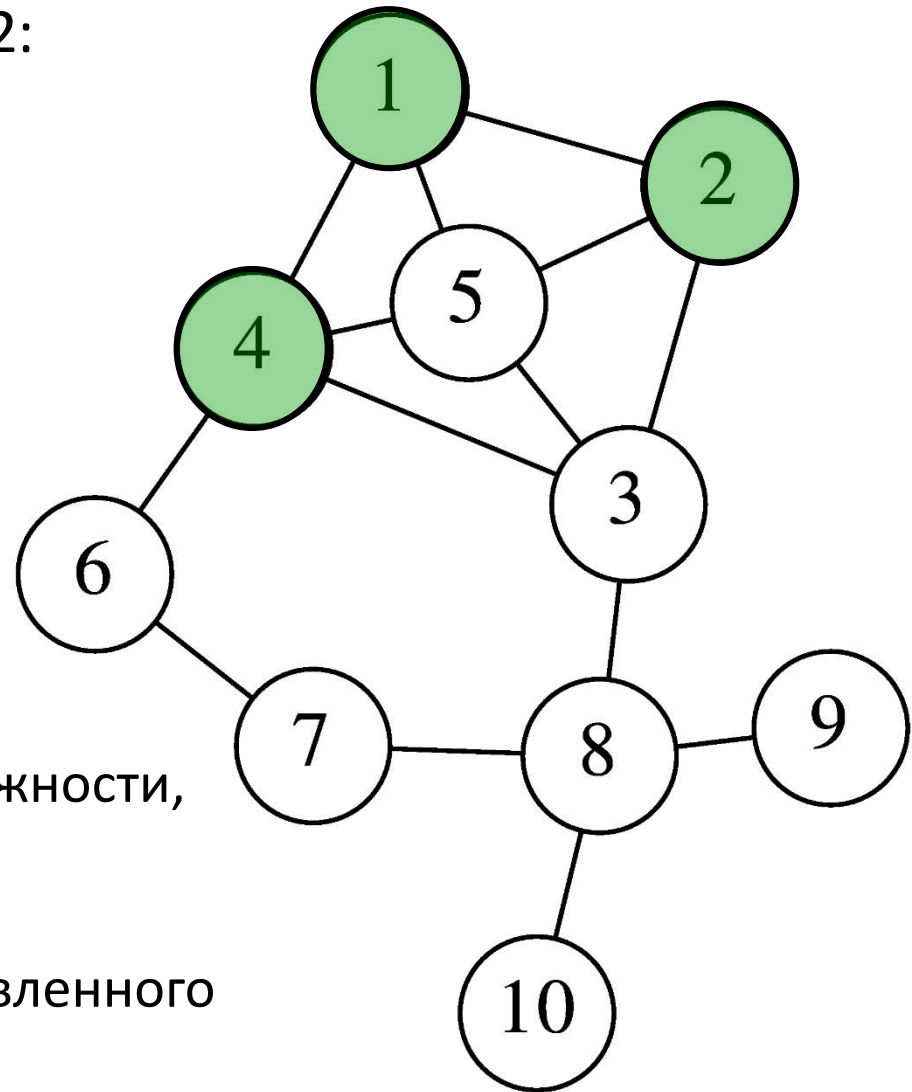


Поиск в глубину (depth-first search)

- Обход в глубину с вершины 2:
DFS(2)

```
for each u in Adj(2) do  
    ...  
end for
```

- Обход в глубину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
- Обход в глубину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$

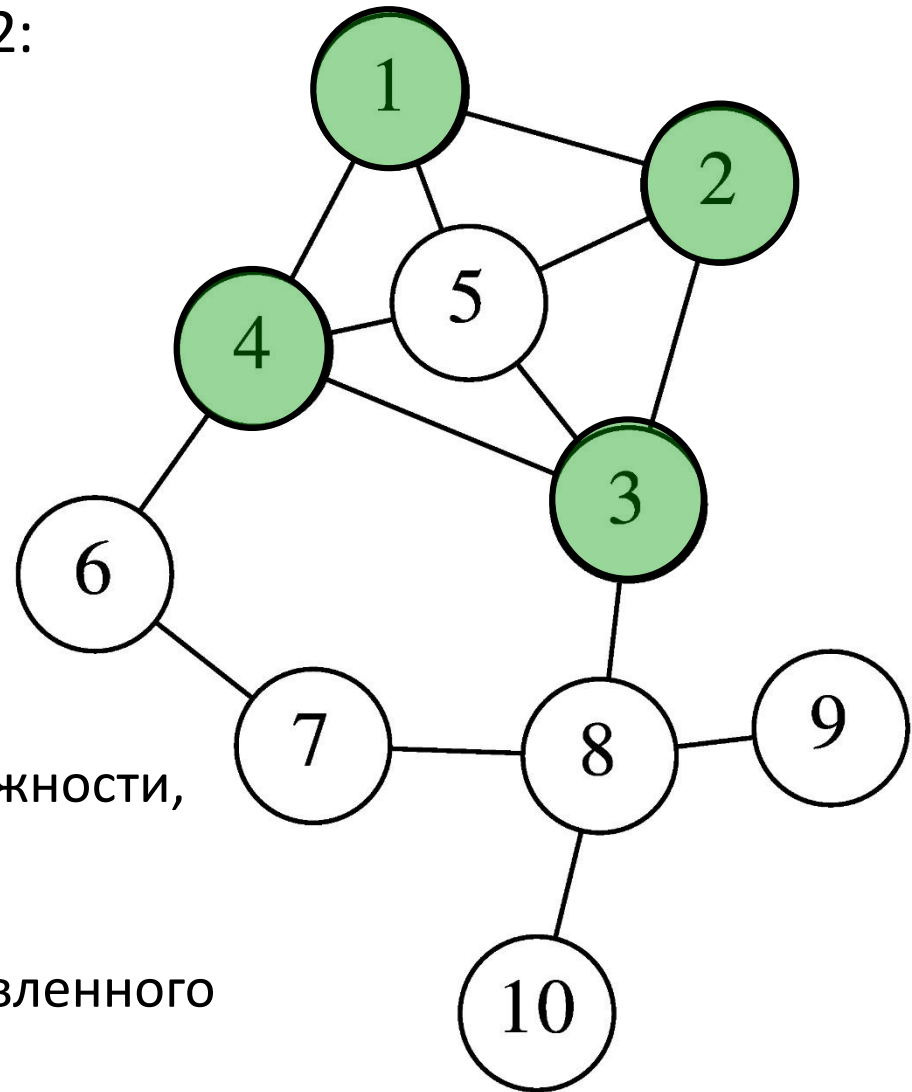


Поиск в глубину (depth-first search)

- Обход в глубину с вершины 2:
DFS(2)

```
for each u in Adj(2) do  
    ...  
end for
```

- Обход в глубину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
- Обход в глубину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$



Поиск в глубину (depth-first search)

- Обход в глубину с вершины 2:
DFS(2)

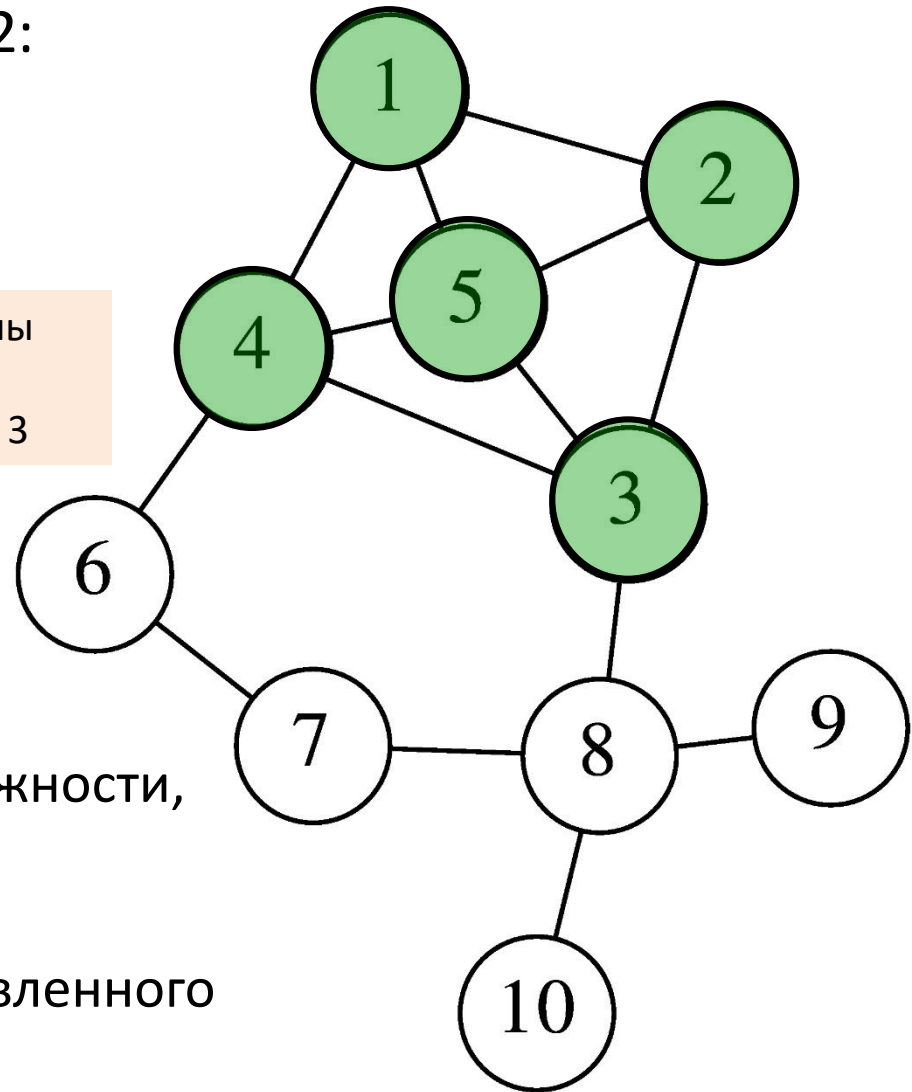
for each u in $\text{Adj}(2)$ do

...

end for

- Все смежные вершины узла 5 посещены
- Возвращаемся к узлу 3

- Обход в глубину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
- Обход в глубину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$

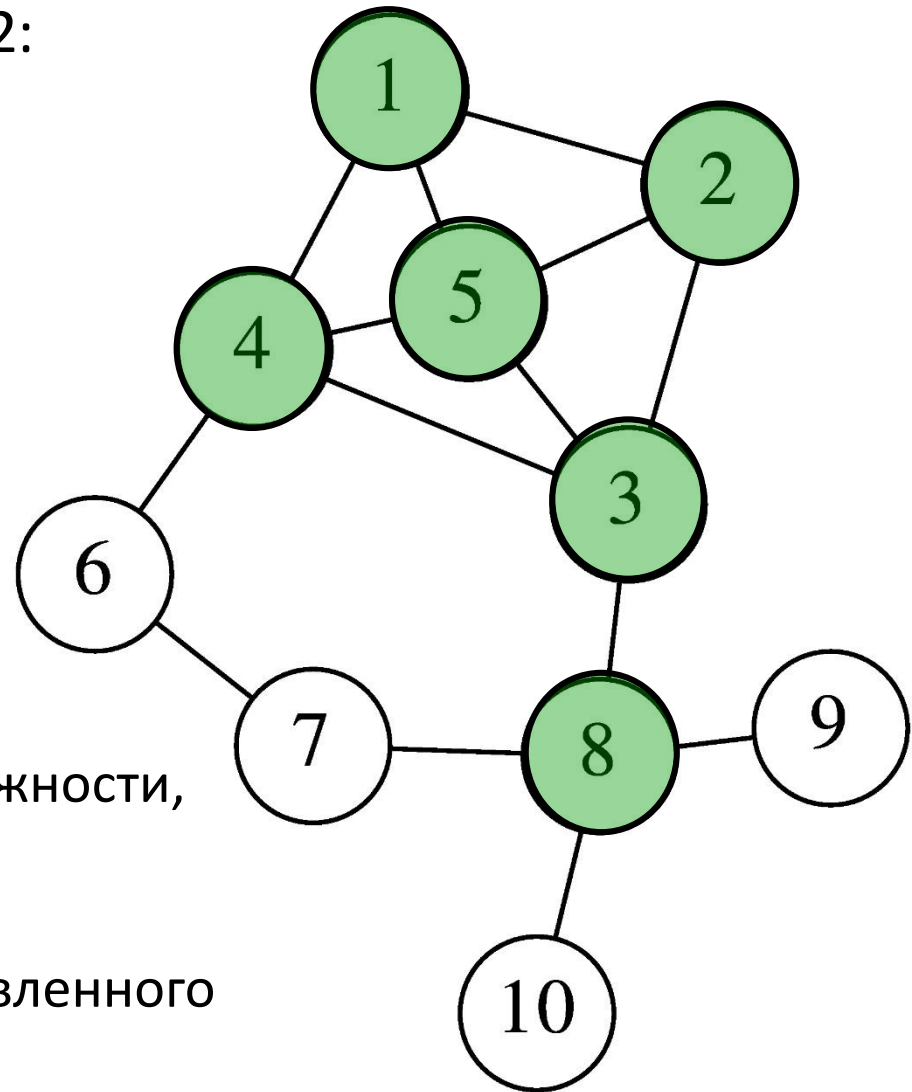


Поиск в глубину (depth-first search)

- Обход в глубину с вершины 2:
DFS(2)

```
for each u in Adj(2) do  
    ...  
end for
```

- Обход в глубину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
- Обход в глубину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$

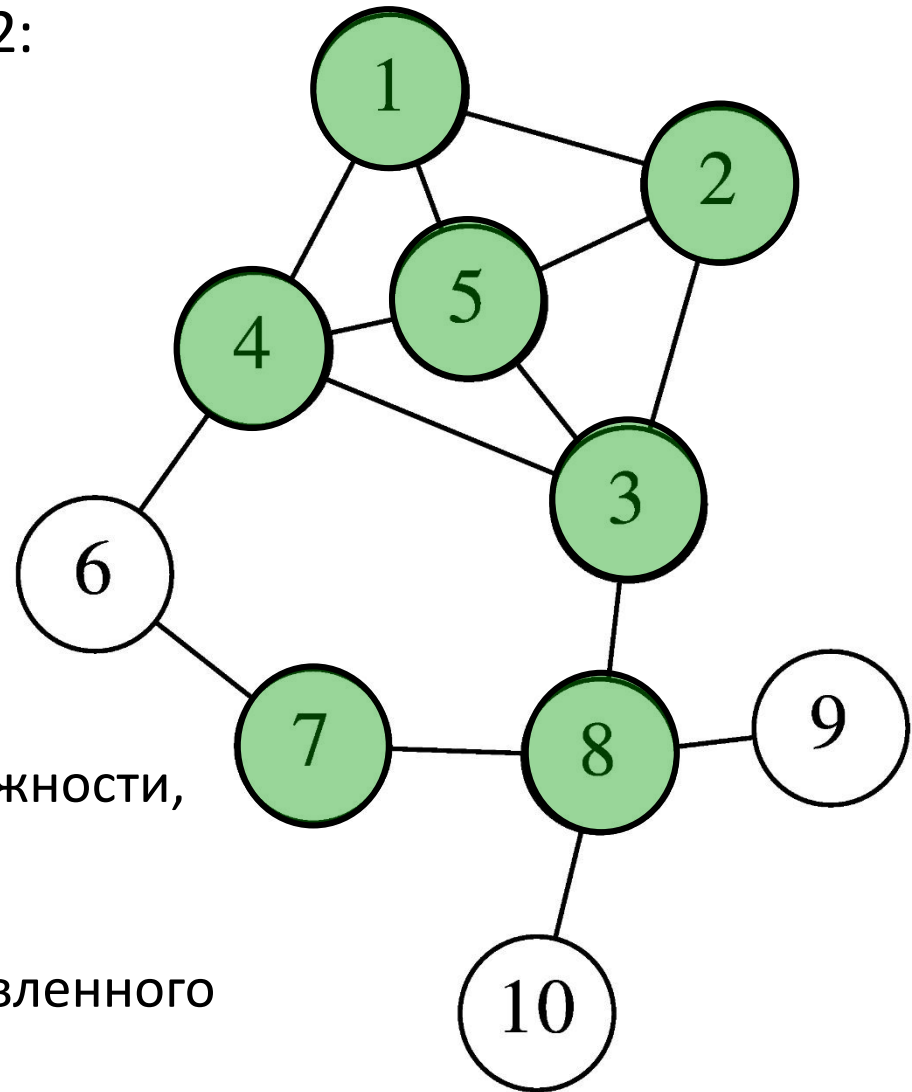


Поиск в глубину (depth-first search)

- Обход в глубину с вершины 2:
DFS(2)

```
for each u in Adj(2) do  
    ...  
end for
```

- Обход в глубину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
- Обход в глубину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$



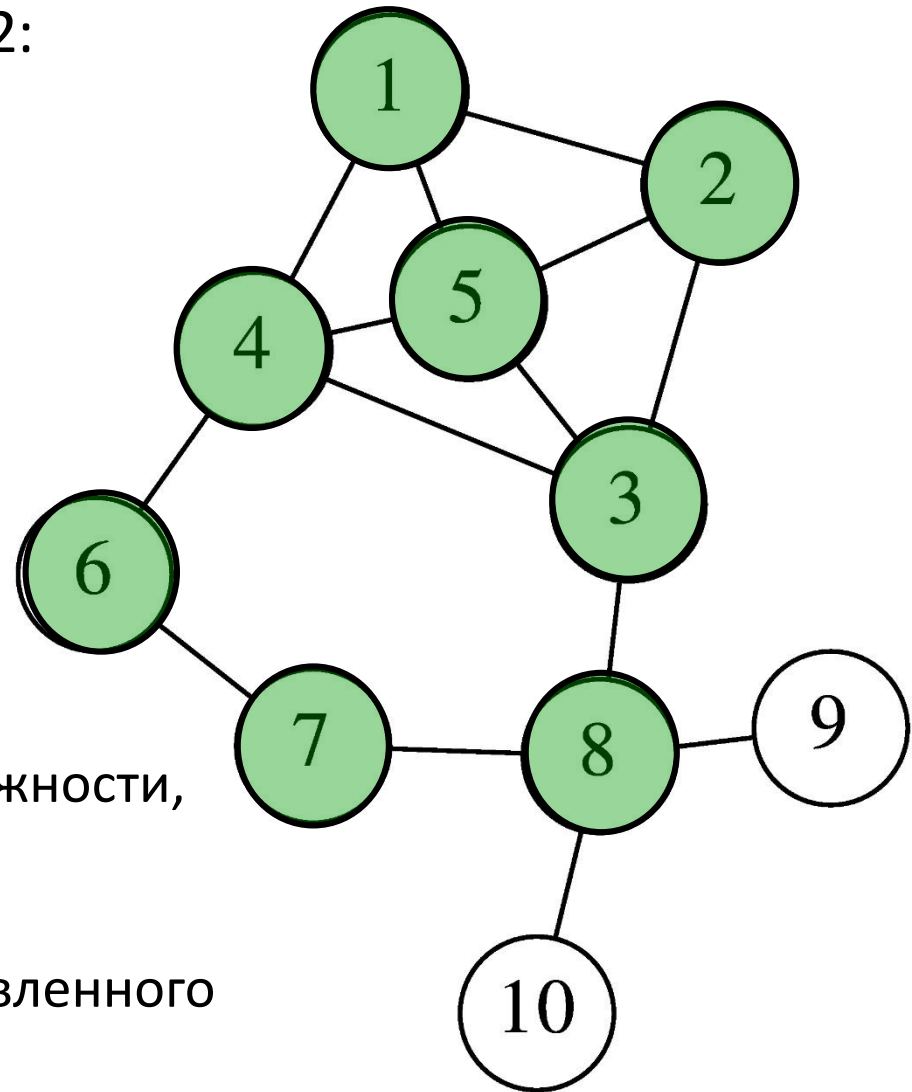
Поиск в глубину (depth-first search)

- Обход в глубину с вершины 2:
DFS(2)

```
for each u in Adj(2) do  
    ...  
end for
```

- Все смежные вершины узла 6
посещены
- Возвращаемся к узлу 7, затем к 8

- Обход в глубину графа,
представленного матрицей смежности,
имеет трудоемкость $O(|V|^2)$
- Обход в глубину графа, представленного
списком смежности, имеет
трудоемкость $O(|V| + |E|)$

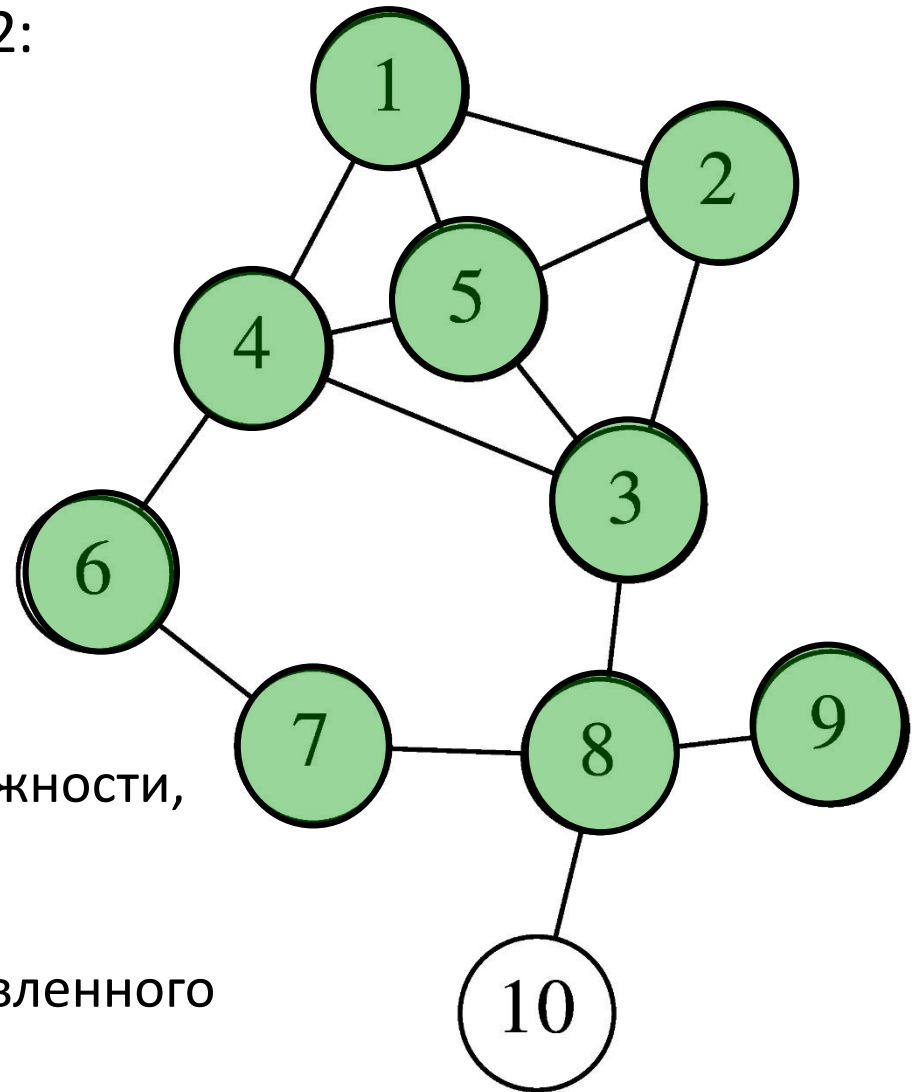


Поиск в глубину (depth-first search)

- Обход в глубину с вершины 2:
DFS(2)

```
for each u in Adj(2) do  
    ...  
end for
```

- Обход в глубину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
- Обход в глубину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$

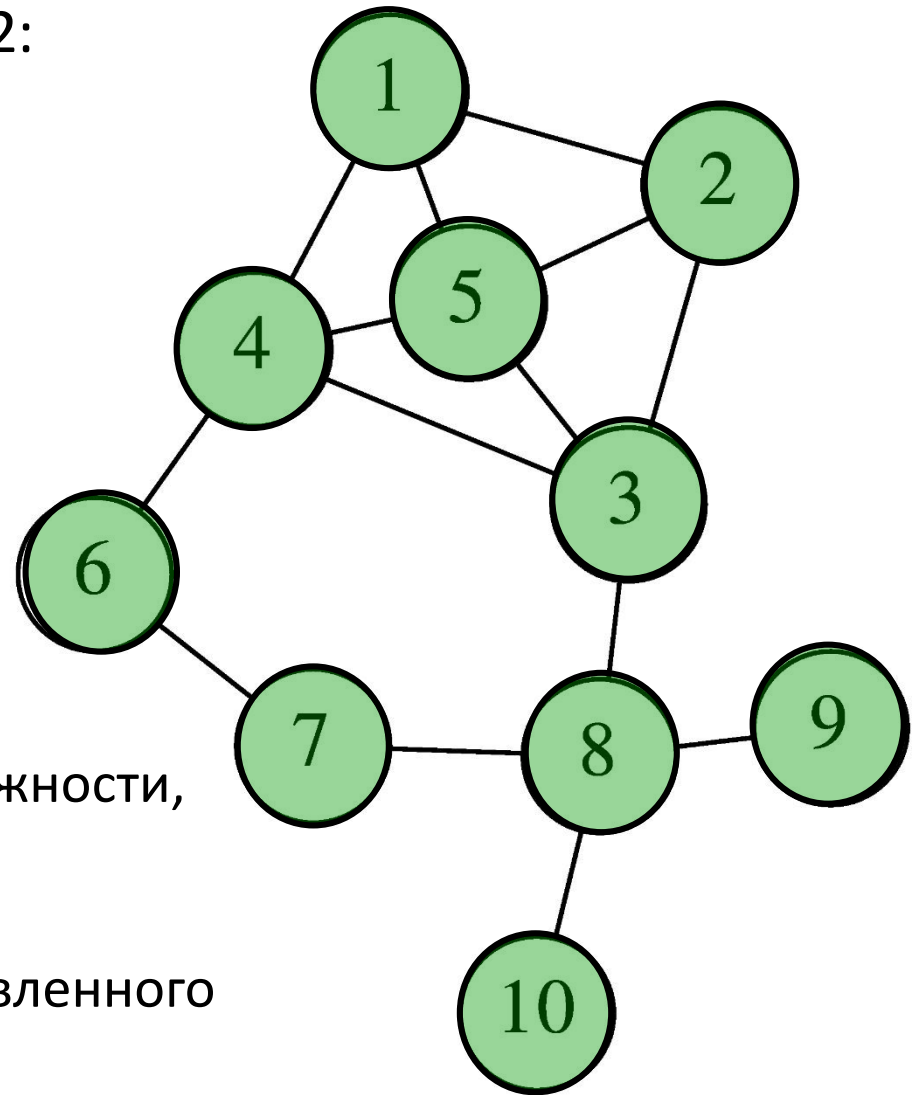


Поиск в глубину (depth-first search)

- Обход в глубину с вершины 2:
DFS(2)

```
for each u in Adj(2) do  
    ...  
end for
```

- Обход в глубину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
- Обход в глубину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$



Поиск в ширину (breadth-first search)

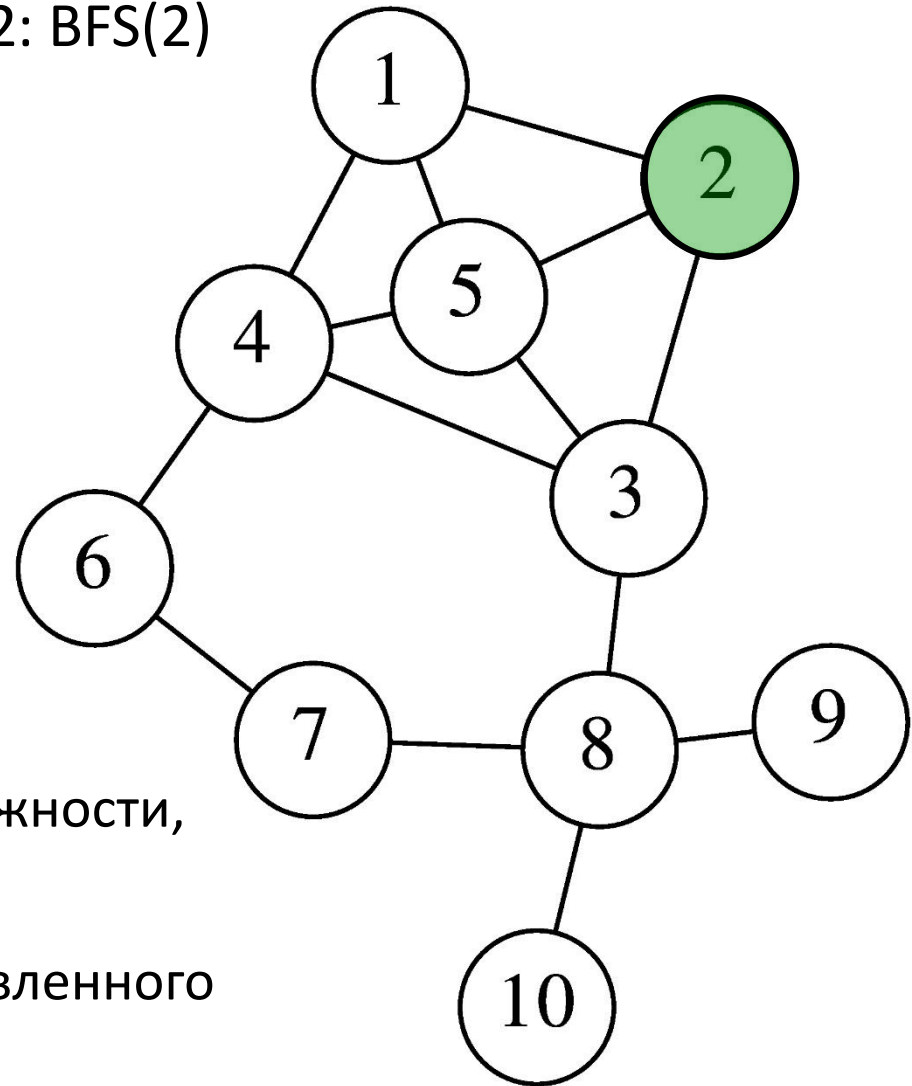
- **Поиск в ширину** (breadth-first search – BFS, обход в ширину) – процедура посещения всех вершин графа начиная с заданного узла v
- Сперва посещаем (обрабатываем) свои дочерние вершины

Поиск в ширину (breadth-first search)

```
function BFS(v)
    visited[v] = true
    // Обрабатываем вершину v
    QueueEnqueue(v)          // Помещаем v в очередь вершин
    while QueueSize() > 0 do
        u = QueueDequeue()   // Извлекаем вершину
        for each x in Adj(u) do
            if visited[x] = false then
                QueueEnqueue(x)
                visited[x] = true
                // Обрабатываем узел x
            end if
        end for
    end while
end function
```

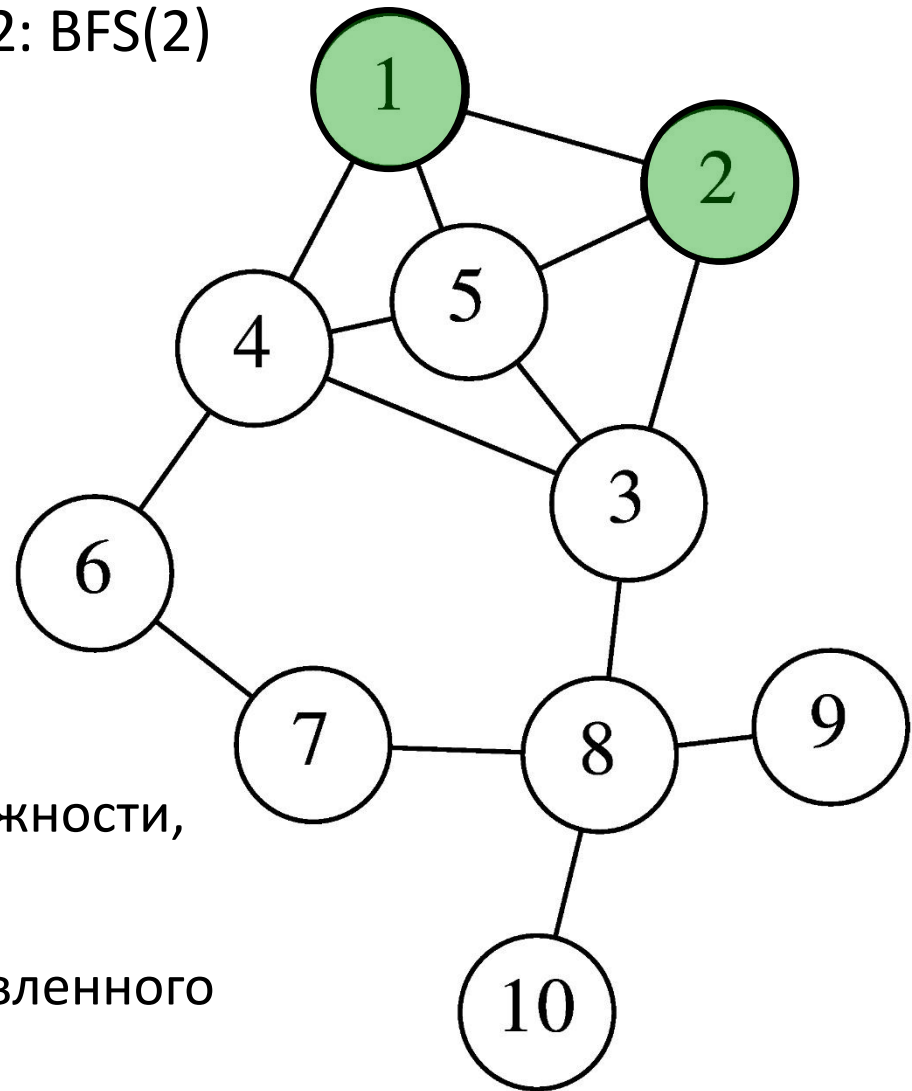
Поиск в ширину (breadth-first search)

- Обход в ширину с вершины 2: BFS(2)
- Извлекли из очереди: **2**
- В очереди:
1, 3, 5
- Обход в ширину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
- Обход в ширину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$



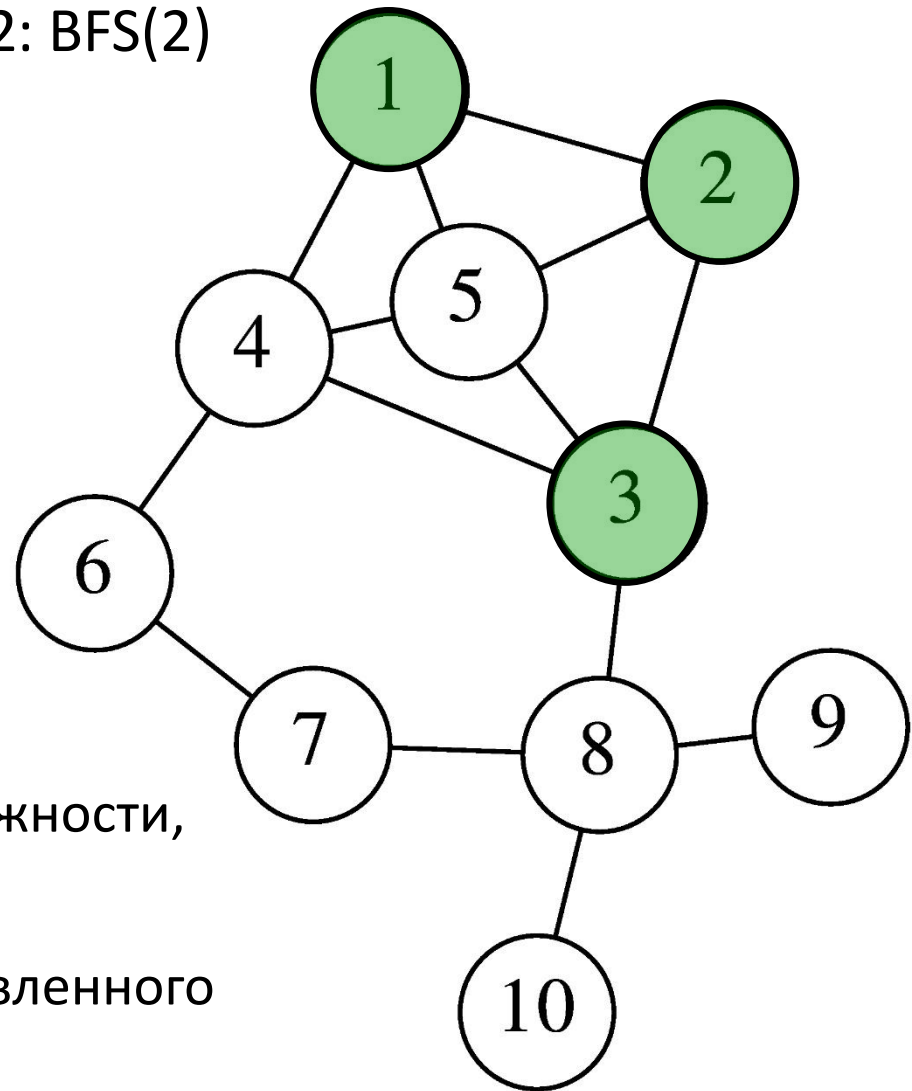
Поиск в ширину (breadth-first search)

- Обход в ширину с вершины 2: BFS(2)
- Извлекли из очереди: **1**
- В очереди:
3, 5, **4**
- Обход в ширину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
- Обход в ширину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$



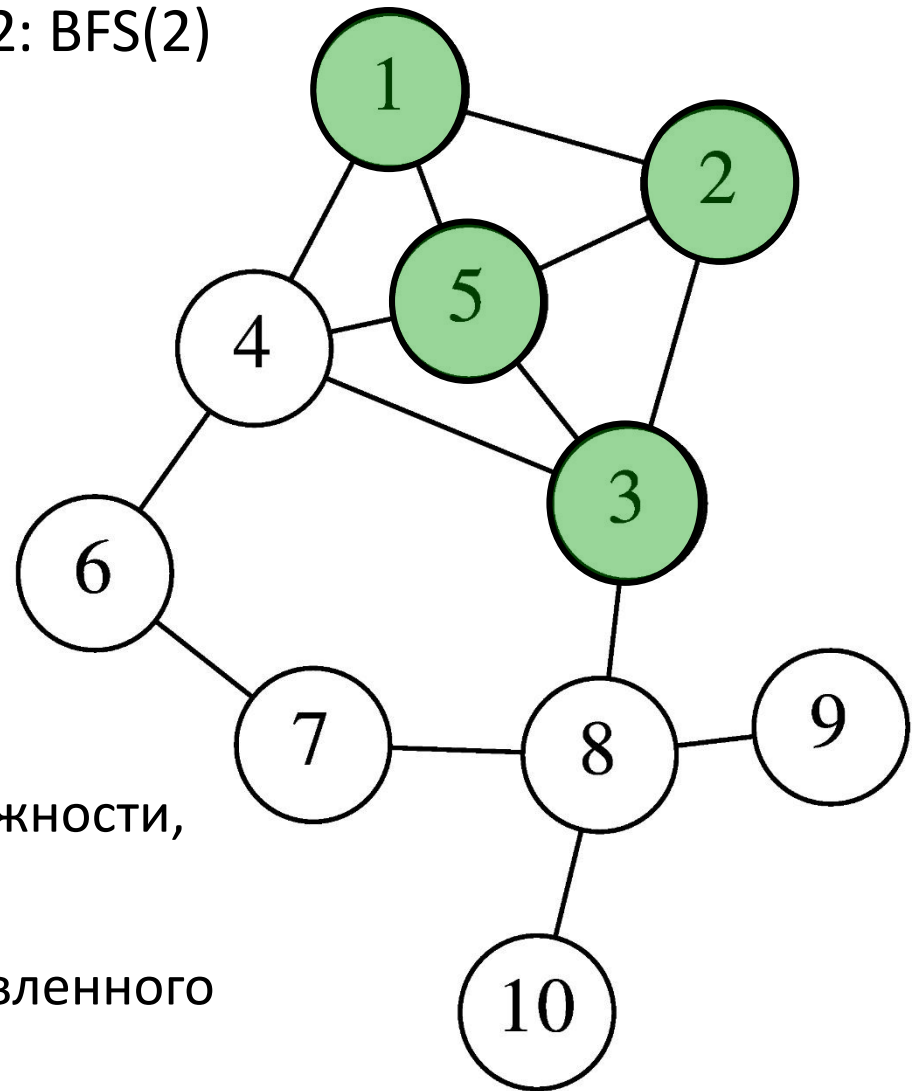
Поиск в ширину (breadth-first search)

- Обход в ширину с вершины 2: BFS(2)
- Извлекли из очереди: **3**
- В очереди:
5, 4, **8**
- Обход в ширину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
- Обход в ширину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$



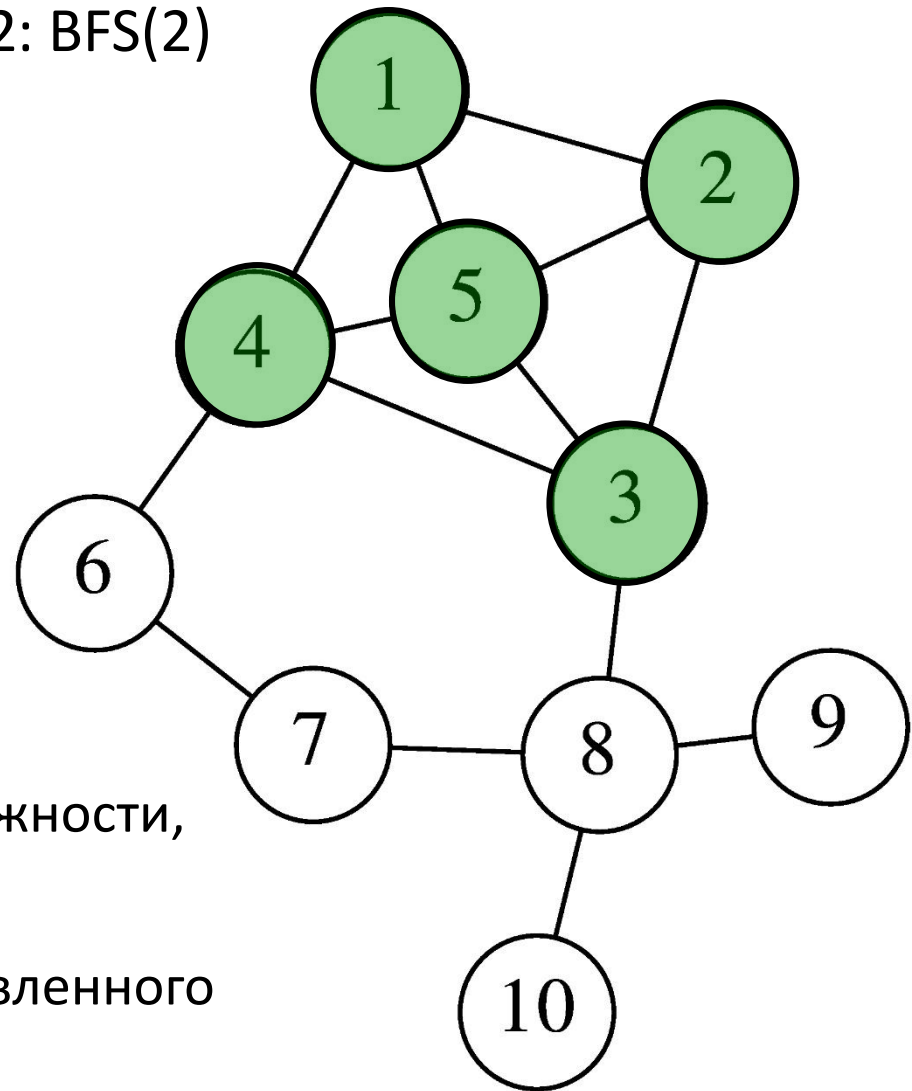
Поиск в ширину (breadth-first search)

- Обход в ширину с вершины 2: BFS(2)
 - Извлекли из очереди: **5**
 - В очереди:
4, 8
-
- Обход в ширину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
 - Обход в ширину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$



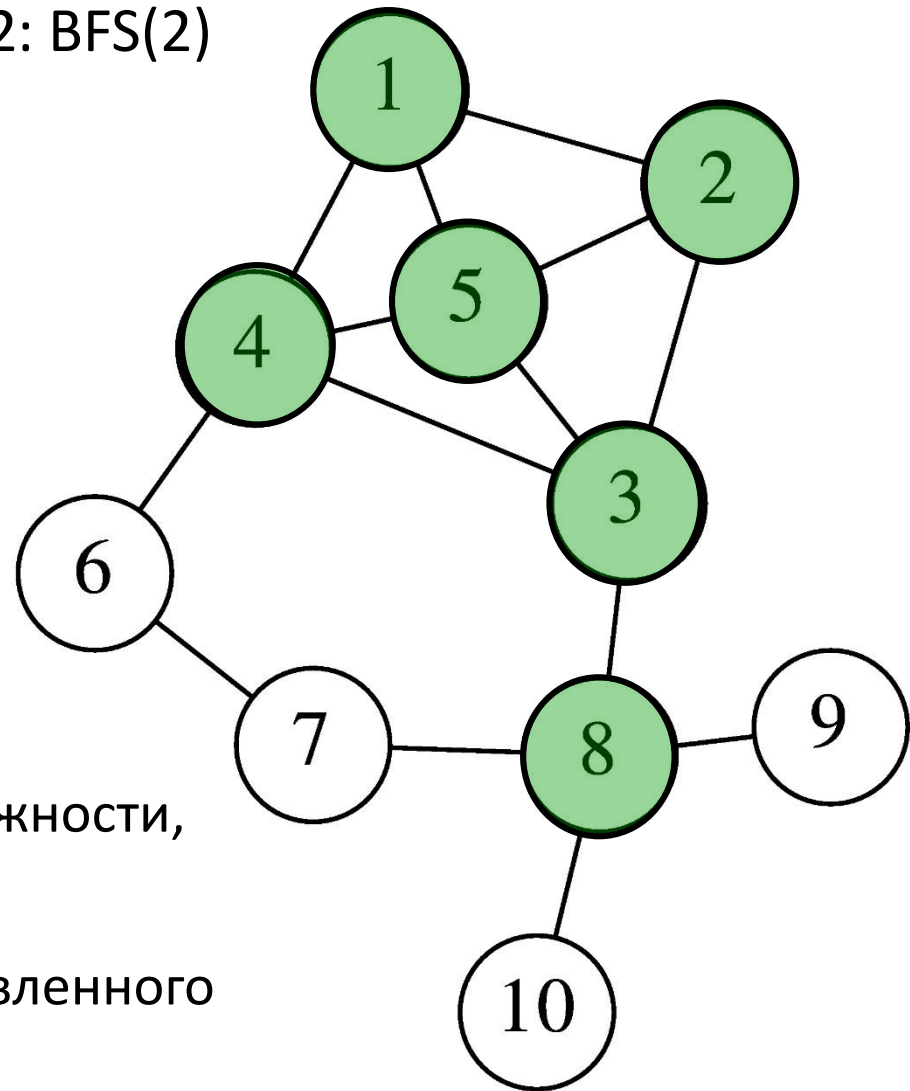
Поиск в ширину (breadth-first search)

- Обход в ширину с вершины 2: BFS(2)
- Извлекли из очереди: **4**
- В очереди:
8, **6**
- Обход в ширину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
- Обход в ширину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$



Поиск в ширину (breadth-first search)

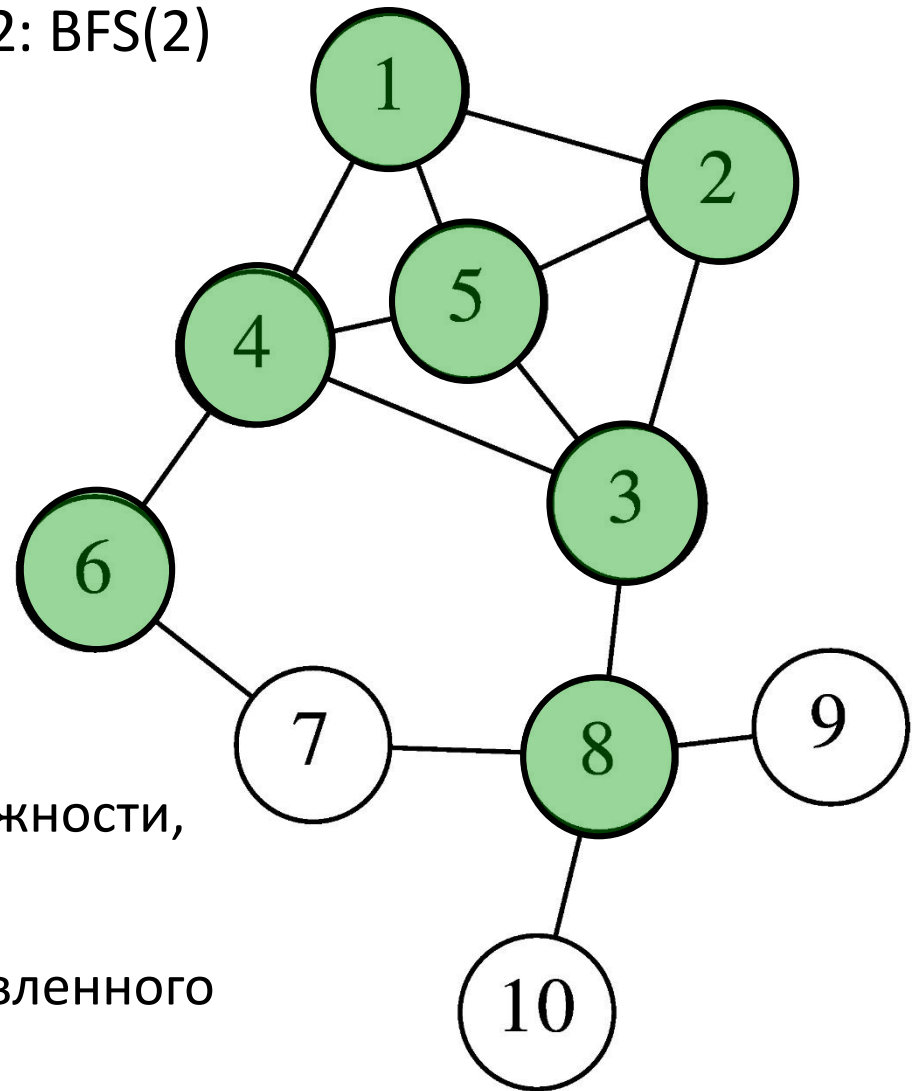
- Обход в ширину с вершины 2: BFS(2)
- Извлекли из очереди: **8**
- В очереди:
6, 7, 9, 10



- Обход в ширину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
- Обход в ширину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$

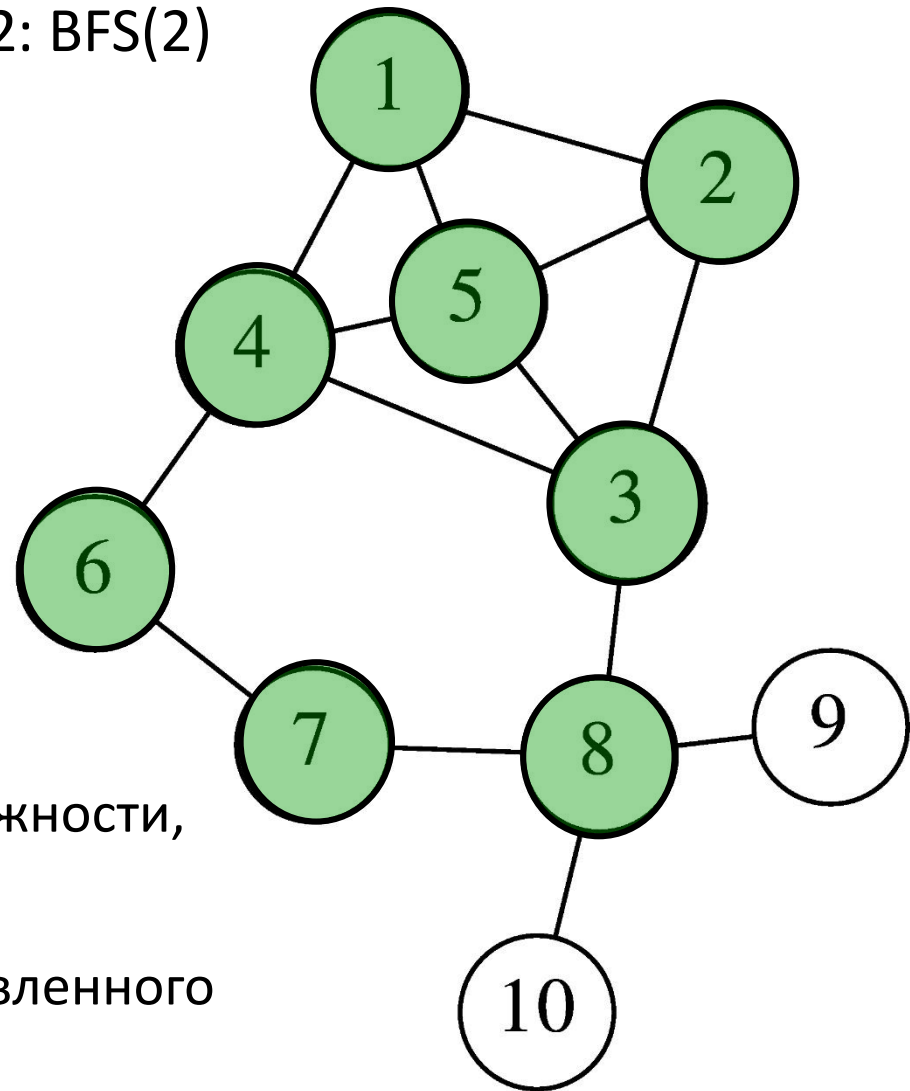
Поиск в ширину (breadth-first search)

- Обход в ширину с вершины 2: BFS(2)
 - Извлекли из очереди: **6**
 - В очереди:
7, 9, 10
-
- Обход в ширину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
 - Обход в ширину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$



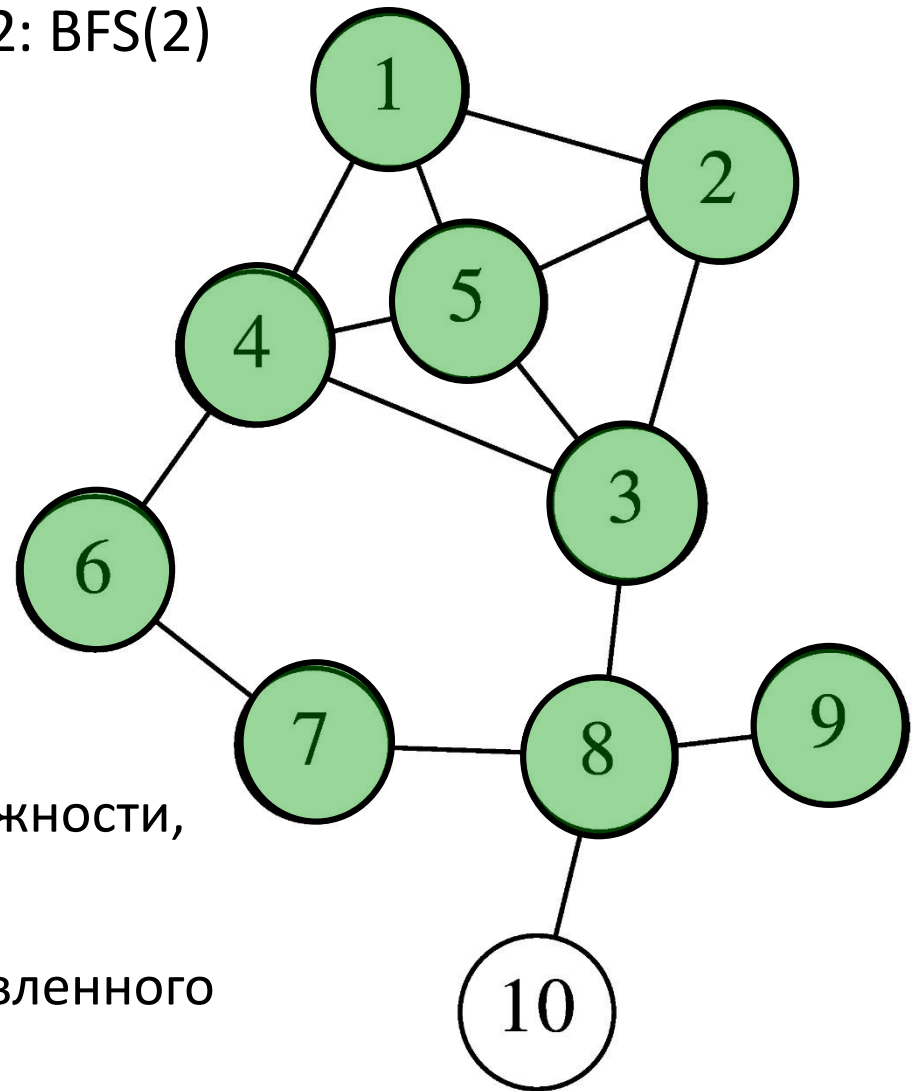
Поиск в ширину (breadth-first search)

- Обход в ширину с вершины 2: BFS(2)
- Извлекли из очереди: **7**
- В очереди:
9, 10
- Обход в ширину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
- Обход в ширину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$



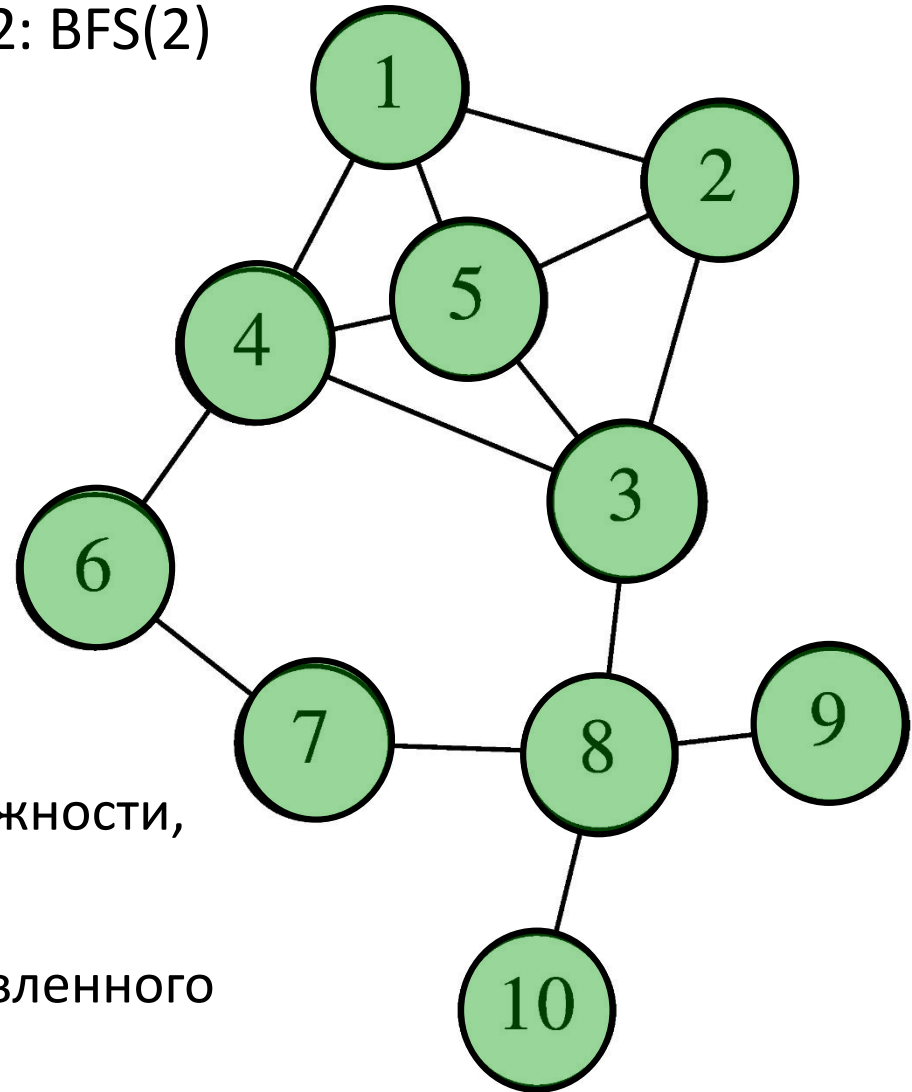
Поиск в ширину (breadth-first search)

- Обход в ширину с вершины 2: BFS(2)
 - Извлекли из очереди: **9**
 - В очереди:
10
-
- Обход в ширину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
 - Обход в ширину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$



Поиск в ширину (breadth-first search)

- Обход в ширину с вершины 2: BFS(2)
 - Извлекли из очереди: **10**
 - В очереди:
-
- Обход в ширину графа, представленного матрицей смежности, имеет трудоемкость $O(|V|^2)$
 - Обход в ширину графа, представленного списком смежности, имеет трудоемкость $O(|V| + |E|)$



Реализация графа на базе матрицы смежности

```
#include <stdio.h>
#include <stdlib.h>

#include "queue_array.h"

struct graph {
    int nvertices;      /* Число вершин */
    int *m;              /* Матрица n x n */
    int *visited;
};
```

Реализация графа на базе матрицы смежности

```
struct graph *graph_create(int nvertices)
{
    struct graph *g;

    g = malloc(sizeof(*g));
    g->nvertices = nvertices;
    g->visited = malloc(sizeof(int) * nvertices);
    g->m = malloc(sizeof(int)
                  * nvertices * nvertices);
    graph_clear(g); // Опционально,  $O(n^2)$ 
    return g;
}
```

$$T_{Create} = O(n^2)$$

$$Memory = O(n^2)$$

Реализация графа на базе матрицы смежности

```
void graph_clear(struct graph *g)
{
    int i, j;

    for (i = 0; i < g->nvertices; i++) {
        g->visited[i] = 0;
        for (j = 0; j < g->nvertices; j++) {
            g->m[i * g->nvertices + j] = 0;
        }
    }
}
```

$$T_{Clear} = O(n^2)$$

Реализация графа на базе матрицы смежности

```
void graph_free(struct graph *g)
{
    free(g->m);
    free(g);
}
```

Реализация графа на базе матрицы смежности

```
/*
 * graph_set_edge: Назначает ребру (i, j) вес w
 *                  i, j = 1, 2, ..., n
 */
void graph_set_edge(struct graph *g,
                    int i, int j, int w)
{
    g->m[(i - 1) * g->nvertices + j - 1] = w;
    g->m[(j - 1) * g->nvertices + i - 1] = w;
}

int graph_get_edge(struct graph *g, int i, int j)
{
    return g->m[(i - 1) * g->nvertices + j - 1];
}
```

Обход графа в глубину (DFS) – рекурсивная версия

```
void graph_dfs(struct graph *g, int v)
{
    int i;

    g->visited[v - 1] = 1;
    printf("Vertex %d\n", v);

    for (i = 0; i < g->nvertices; i++) {
        if (g->m[(v - 1) * g->nvertices + i] > 0
            && g->visited[i] == 0)
        {
            graph_dfs(g, i + 1);
        }
    }
}
```

$$T_{DFS} = O(n^2)$$

Обход графа в ширину (BFS)

```
void graph_bfs(struct graph *g, int v)
{
    int i, j;
    struct queue *q;

    for (i = 0; i < g->nvertices; i++)
        g->visited[i] = 0;

    // Создаем очередь
    q = queue_create(g->nvertices);

    // Обрабатываем стартовую вершину
    g->visited[v - 1] = 1;
    printf("Vertex %d\n", v);
    queue_enqueue(q, v - 1);
}
```

Обход графа в ширину (BFS, продолжение)

```
while (queue_size(q) > 0) {
    i = queue_dequeue(q);
    for (j = 0; j < g->nvertices; j++) {
        if (g->m[i * g->nvertices + j] > 0
            && g->visited[j] == 0)
        {
            queue_enqueue(q, j);
            g->visited[j] = 1;
            printf("Vertex %d\n", j + 1);
        }
    }
}
queue_free(q);
}
```

$$T_{BFS} = O(n^2)$$

Пример

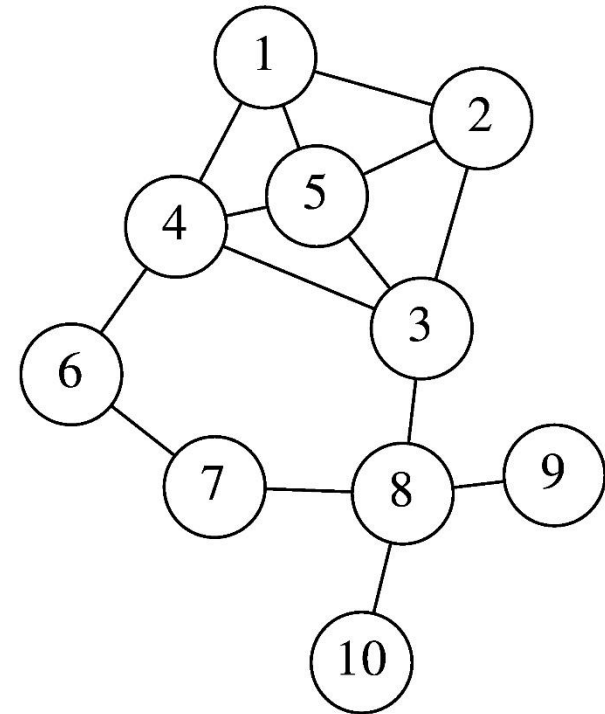
```
int main()
{
    struct graph *g;

    g = graph_create(10);
    graph_set_edge(g, 1, 2, 1);
    graph_set_edge(g, 1, 4, 1);
    /* ... */

    printf("DFS:\n");
    graph_dfs(g, 2);

    printf("BFS:\n");
    graph_bfs(g, 2);

    graph_free(g);
    return 0;
}
```



Домашнее чтение

- Разобрать доказательство вычислительной сложности процедур BFS и DFS [Aho, С. 217], [CLRS, С. 630, С. 639]
- Прочитать про формат хранения графов в сжатом виде
 - ❑ Compressed Sparse Row Graph (CSR)
http://www.boost.org/doc/libs/1_45_0/libs/graph/doc/compressed_sparse_row.html
 - ❑ Sparse matrix
http://en.wikipedia.org/wiki/Sparse_matrix