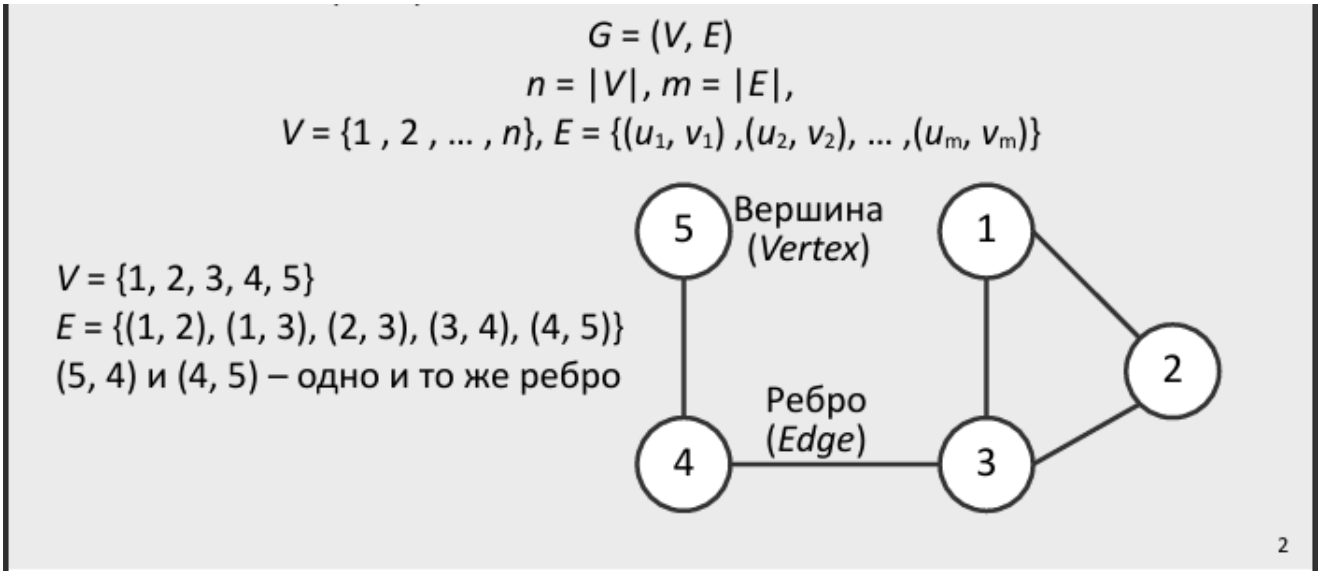
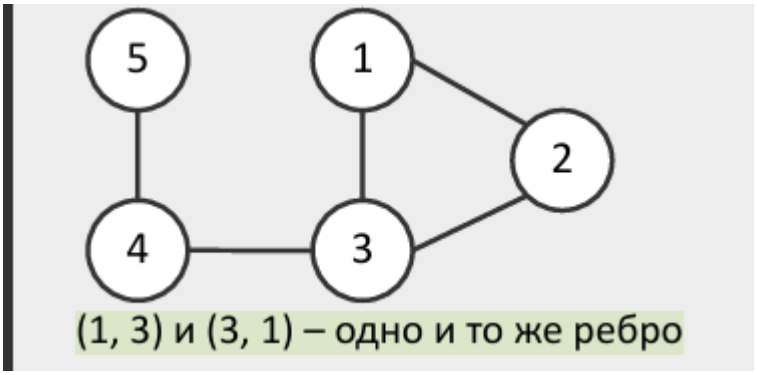
Графы. Виды графов. Способы представления графов в памяти. Реализация графа на основе матрицы смежности.

Графы

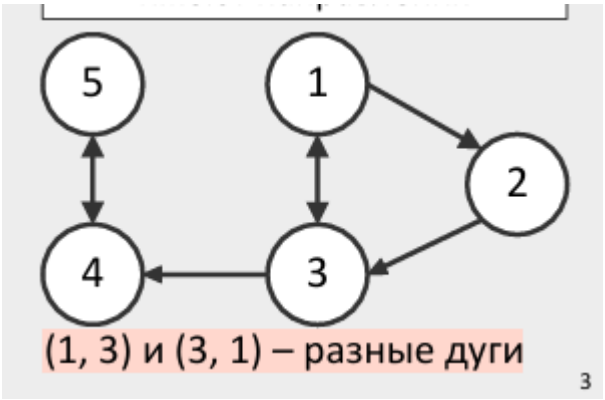
● Граф (graph) – это совокупность непустого множества V вершин и множества E ребер



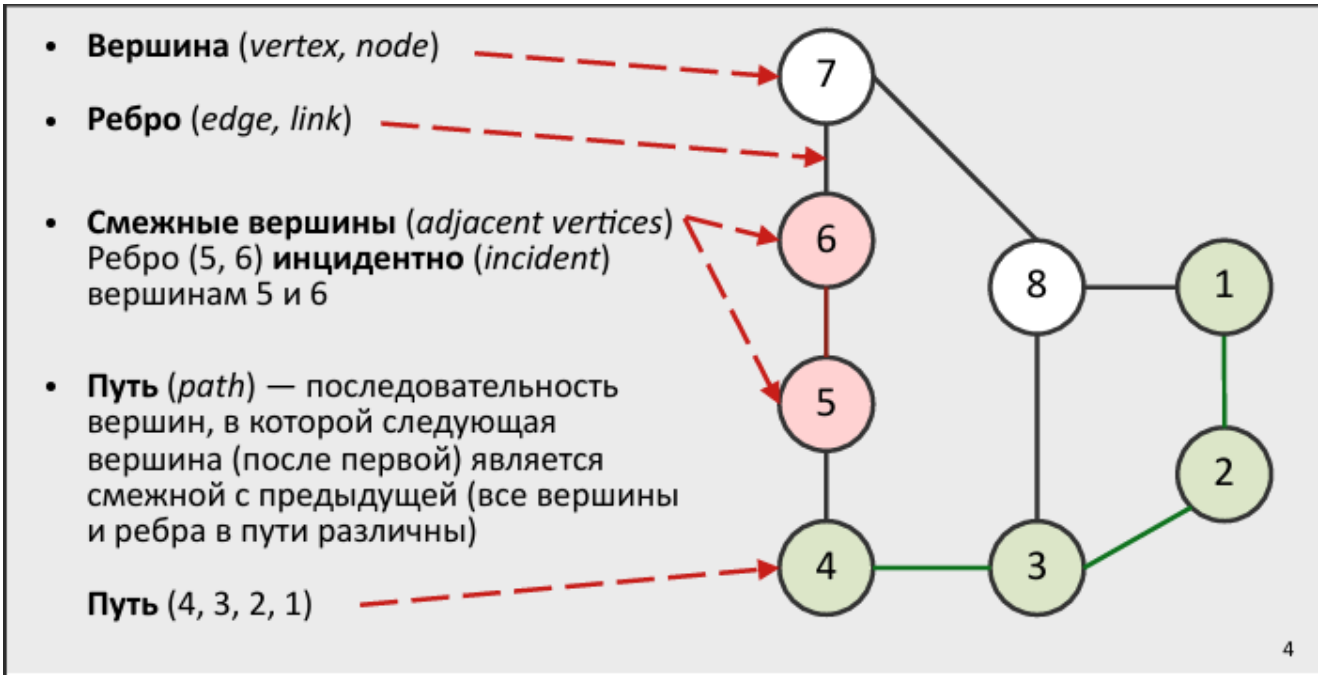
Виды графов

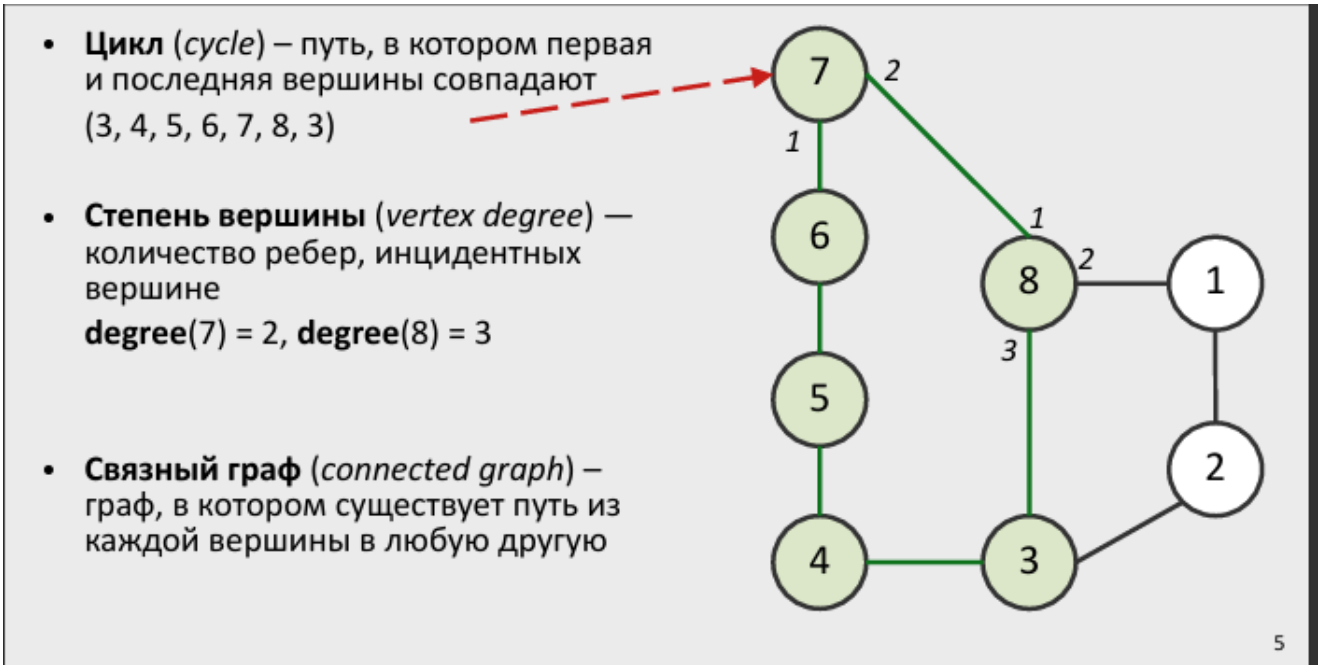
- **Неориентированные графы** *(undirected graphs)*
- **Ребра** *(edges)* не имеют направлений
- 

- Ориентированные графы *(directed graphs)*
- **Ребра** – **дуги** (arcs, edges) имеют направления



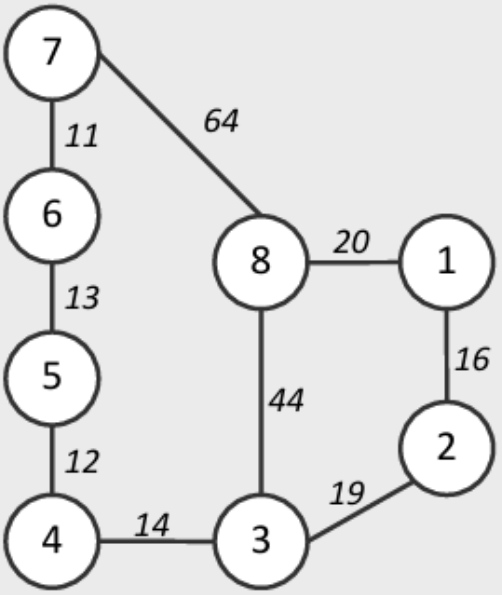
Основные определения





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

- $w_{12} = 16$
- $w_{23} = 19$
- $w_{34} = 14$
- $w_{38} = 44$



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

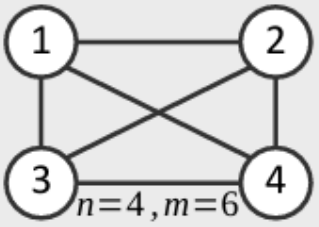
Количество ребер в полном неориентированном графе:

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

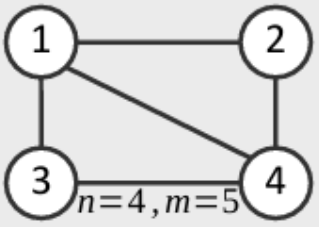
Насыщенность D графа (*density*):

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

У полного графа насыщенность D = 1



$$D = \frac{2 \cdot 6}{4 \cdot (4-1)} = \frac{12}{12} = 1$$

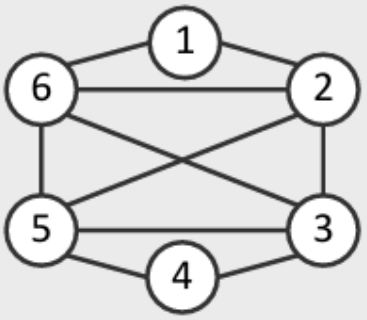


$$D = \frac{2 \cdot 5}{4 \cdot (4-1)} = \frac{10}{12} = 0.83$$

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

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

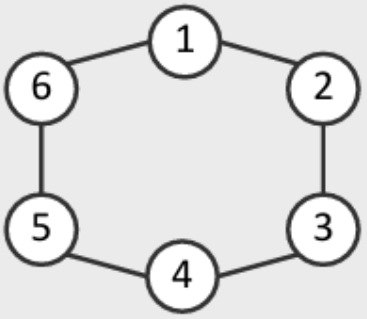
$$D=\frac{2\cdot 10}{6\cdot(6-1)}=\frac{20}{30}=0.67,\quad D>0.5$$



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

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

$$D=\frac{2\cdot 6}{6\cdot(6-1)}=\frac{12}{30}=0.4,\quad D<0.5$$



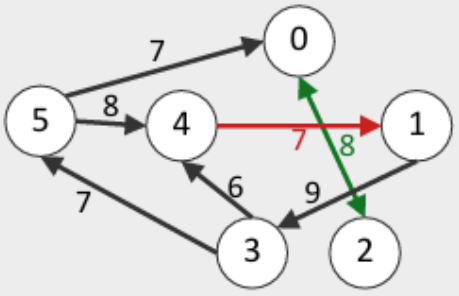
8

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
  Список смежности *(adjacency list)* – эффективен для разреженных графов

Матрица смежности

	0	1	2	3	4	5
0	0	0	8	0	0	0
1	0	0	0	9	0	0
2	8	0	0	0	0	0
3	0	0	0	0	6	7
4	0	7	0	0	0	0
5	7	0	0	0	8	0

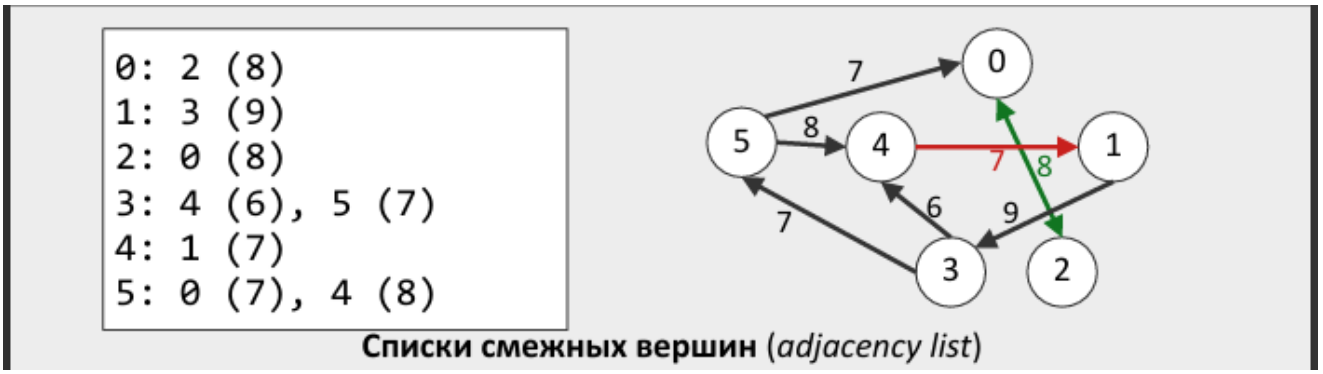


Матрица смежности (*adjacency matrix*)

- Хранение в массиве
- Память $O(|V|^2)$
-
- Доступ к ребру A [i,j]: **O(1)**

- Эффективна для насыщенных графов $(|E| \approx |V|^2)$

Список смежных вершин



- Хранение списков в массиве указателей
- Память O(|E| + |V|)
- Эффективен для разреженных графов (|E| ≈ |V|)
- Доступ к ребру A[i , j]: **O(|V|)**
- Доступ к ребру A[i , j]: номер вершины i – номер связного списка, в котором необходимо найти элемент с ключом j, вес ребра – значение

Матрица смежности (*adjacency matrix*)

- V·V·sizeof(datatype)
  - 100·100·sizeof(int) = 40000 байт

Списки смежных вершин (*adjacency list*)

- E·(sizeof(datatype) + sizeof(int) + next):
  - (150)·(sizeof(int) + sizeof(int) + 8) = 2400 байт

В данном случае хранение графа с помощью матрицы смежности требует в 40000 / 2400 = 16.67 раз больше памяти

15

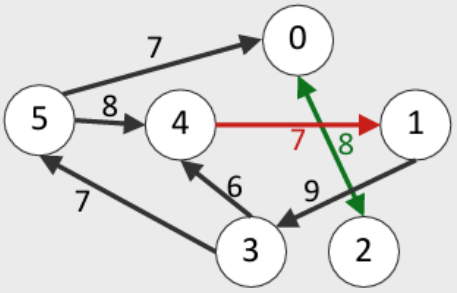
Сжатое хранение строкой (Compressed Sparse Row, CSR)

0	1	2	3	4	5	6
0	1	2	3	5	6	8

A[n + 1]

0	1	2	3	4	5	6	7
2	3	0	4	5	1	0	4
8	9	8	7	6	7	7	8

L[m]



- Память O(|E| + |V|)
- Доступ к ребру a_{ij}: O(|V|)
- Количество смежных узлов вершины i: A[i + 1] – A[i]
- Также возможно **сжатое хранение столбцом** (*Compressed Sparse Column, CSC*)

2 массива:

1. Массив вершин, индексы соответствуют номерам вершин, данные – началу ребер в списке смежных вершин
2. Список смежных вершин содержит вершину и длину пути

16

Списки смежных вершин (*adjacency list*)

E·(sizeof(datatype) + sizeof(int) + next)

150·(sizeof(int) + sizeof(int) + 8) = 2400 байт

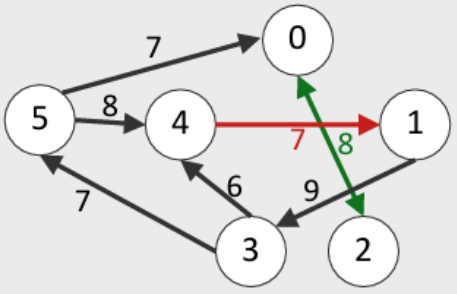
Сжатое хранение строкой (*Compressed Sparse Row*)

(V+1)·sizeof(datatype)+E·(sizeof(datatype)+sizeof(datatype))

101·sizeof(int) + 150·(sizeof(int) + sizeof(int)) = 1604 байт

Список координат

0	1	2	3	4	5	6	7	
0	1	2	3	3	4	5	5	i
2	3	0	4	5	1	0	4	j
8	9	8	7	6	7	7	8	w

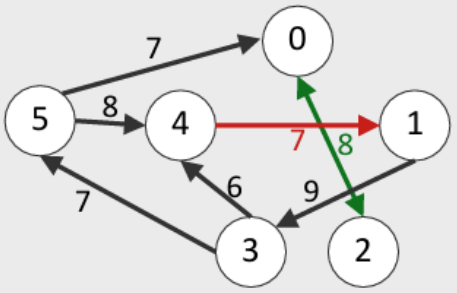


3 массива: строка, столбец и значение

- Нам нужно получить всю строку или столбец
- Предположим, что для первой части программы нам нужно будет извлекать строки, а для второй – столбцы
- Из списка координат можно перейти в формат сжатого хранения строкой или столбцом

25

0	1	2	3	4	5	6	7	
0	1	2	3	3	4	5	5	*i*
2	3	0	4	5	1	0	4	*j*
8	9	8	7	6	7	7	8	*w*



3 массива: строка, столбец и значение

Доступ к ребру a_{ij}: O(|*E*|), необходимо сначала найти вершину *i*

E·sizeof(datatype)·2 + E·sizeof(datatype)

8·sizeof(int)·2 + 8·sizeof(int) = 96 байт

26

Реализация графа на основе матрицы смежности

```c
#include "queue_array.h"
struct graph {
    int nvertices; /* Число вершин */
    int *m; /* Матрица n x n */
    int *visited;
};
```

Создание графа

```c
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;
}
```

Очистка и удаление графа

```c
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;
            }
        }
    }

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

```
/* * 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];
}
```