# Графы

*Баев А.Ж.*

## Содержание

# 1 Структуры данных: матрица смежности и список смежных ребер

**Постановка задачи.** Построить матрицу смежности и список смежных ребер для данного графа.

**Структуры данных.** Дан неориентированный граф без кратный ребер и петель. Вершины графа пронумерованы от 1 до $n$.

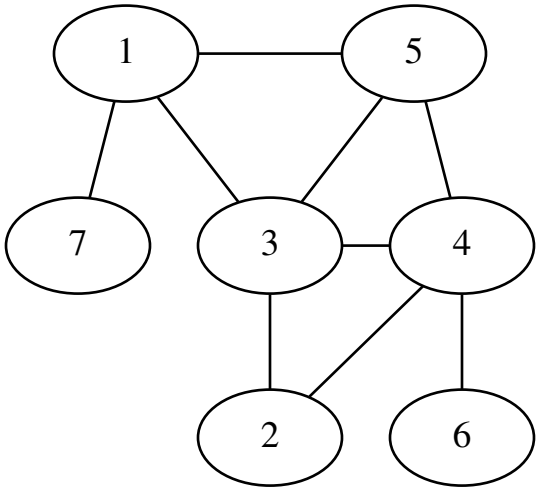


Fig. 1: Пример графа.

Матрица смежности (инцендентности) определяется следующим образом:

$$m_{ij} = \begin{cases} 1, \text{если между вершинами } i \text{ и } j \text{ есть ребро,} \\ 0, \text{если между вершинами } i \text{ и } j \text{ нет ребра.} \end{cases}$$

Для данного примера матрица смежности выглядит следующим образом:

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

Список смежных ребер определяется следующим образом: $l_{ik}$ — $k$-я по счету «соседняя» вершина. Порядок нумерации таких вершин не важен.

Для данного примера матрица смежности выглядит следующим образом:

| количество | соседи | | | |
|---|---|---|---|---|
| 3 | 3 | 5 | 7 | |
| 2 | 3 | 4 | | |
| 4 | 1 | 2 | 4 | 5 |
| 4 | 2 | 3 | 5 | 6 |
| 3 | 1 | 3 | 4 | |
| 1 | 4 | | | |
| 1 | 1 | | | |

**Асимптотика.** Оценим объем памяти, требуемый для хранения с помощью данных структур графа из $n$ вершин и $m$ ребер. Будем считать, что одно целое число занимает одну ячейку памяти. Матрица смежности не зависит от количества ребер: $n^2$ ячеек памяти. Список смежных ребер: $n + m$ ячеек памяти. Заметим, что в полном графе $m = \frac{n(n-1)}{2}$, поэтому список смежных ребер выгоден для разреженных графов.

**Практика.** Даны 2 целых числа: $n$ — количество вершин графа и $m$ — количество ребер. Далее $m$ пар чисел $v_i$ и $w_i$ — номера вершин, соответствующие $i$-му ребру. Вывести список смежных ребер и матрицу смежности.

## 2 Рекурсивный обход в глубину

Дан неориентированный граф без кратный ребер и петель. Вывести все вершины графа при обходе «в глубину».

**Метод решения.** Обход в глубину (англ. depth first search) позволяет перебрать все вершины связного графа в определенном порядке.

Для обхода понабодится дополнительный массив флагов $used$, в котором отмечаются посещенные вершины:

$$used[v] = \begin{cases} 1, \text{если вершина } v \text{ уже посещена,} \\ 0, \text{если вершина } v \text{ еще не посещена.} \end{cases}$$

Изначальновсе вершины считаются не посещенным.

Для текущей вершины переберем всех соседей (в нашей реализации всех по возрастанию номеров). Выберем из них тех, кто еще не был посещен, и перейдем к ним.

**Пример.** Начнем обход в глубину графа на рисунке 3 с вершины 1. Рядом с каждым действием для наглядности написано состояние массива $used$.

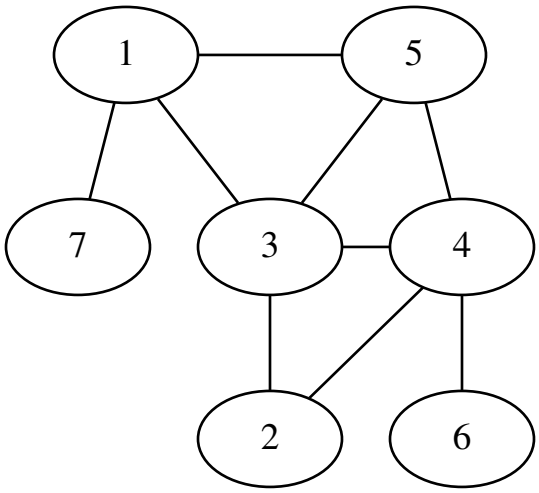


Fig. 2: Пример графа.

**Вершина 1 — посещена.** [1, 0, 0, 0, 0, 0, 0]
Ищем непосещенного соседа 1: находим 3.
**Вершина 3 — посещена.** [1, 0, 1, 0, 0, 0, 0]
Ищем непосещенного соседа 3: находим 2.
**Вершина 2 — посещена.** [1, 1, 1, 0, 0, 0, 0]
Ищем непосещенного соседа 2: находим 4.
**Вершина 4 — посещена.** [1, 1, 1, 1, 0, 0, 0]
Ищем непосещенного соседа 4: находим 5.
**Вершина 5 — посещена.** [1, 1, 1, 1, 1, 0, 0]
Ищем непосещенного соседа 5: нет.
Ищем непосещенного соседа 4: находим 6.
**Вершина 6 — посещена.** [1, 1, 1, 1, 1, 1, 0]
Ищем непосещенного соседа 6: нет.
Ищем непосещенного соседа 4: нет.
Ищем непосещенного соседа 2: нет.
Ищем непосещенного соседа 3: нет.
Ищем непосещенного соседа 1: находим 7.
**Вершина 7 — посещена.** [1, 1, 1, 1, 1, 1, 1]
Ищем непосещенного соседа 7: нет.
Ищем непосещенного соседа 1: нет.

**Псевдокод.** Для матрицы смежности `incendMatrix`.

```
dfs(v)
    used[v] := 1
    print(v)
    for next := 1..n do
        if (incendMatrix[v][next] = 1) and (used[next] = 0) then
            dfs(next)
```

Для списка смежных рёбер `incendList` (количество соседей v-й вершины хранится в `incendList[v][0]`).

```
dfs(v)
    used[v] := 1
    print(v)
    for i := 1..incendList[v][0] do
        next := incendList[v][i]
        if (used[next] = 0) then
            dfs(next)
```

**Асимптотика.** Удобно считать за действие количество операций сравнений `used[next] = 0`. В худшем случае при обходе в глубину с матрицей смежности будут посещены все $n$ вершин и из каждой будут сделаны $n$ таких проверок. Итог: $\Theta\left(n^3\right)$. В худшем случае при обходе в глубину со списком смежных ребер будут сделаны переходы по всем ребрам. Итог: $\Theta\left(m\right)$.

**Практика.** Даны 2 целых числа: $n$ — количество вершин графа и $m$ — количество ребер. Далее $m$ пар чисел $v_i$ и $w_i$ — номера вершин, соответствующие $i$-му ребру. Вывести порядок вершин при обходе данного графа в глубину. Сравнить обход при матрице смежности и при списке смежных ребер.

## 3 Нерекурсивный обход в глубину и обход в ширину

Дан неориентированный граф без кратный ребер и петель. Вывести все вершины графа при обходе «в глубину».

**Метод решения.** Обход в глубину (англ. depth first search) можно реализовать без рекурсии, если использовать дополнительный стек для вершин (фактически при рекурсивном обходе в глубину эту роль выполняет системный стек). Стек — структура данных, которая позволяет добавлять элементы в него, или извлекать элементы из него. Причем извлекается всегда последний добавленный элемент.

Стек можно смоделировать на динамическом или статическом массиве *stack* и указателем на последний добавленный элемент *tail*. Во втором случае, необходимо заранее знать максимальную возможную заполненность стека. В нашем случае размер стека не может превосходить количества вершин графа.

Изначально все вершины считаются не посещенным. Добавляем первую вершину в стек. Далее повторяем следующие действия: если стек не пуст, вытаскиваем текущую вершину, и добавляем в стек **первого** непосещенного соседа.

Обход в ширину(англ. breadth first search) можно реализовать только без рекурсии, если использовать дополнительную очередь. Очередь — структура данных, которая позволяет добавлять элементы в него, или извлекать элементы из него. Причем извлекается всегда первый добавленный элемент.

Очередь можно смоделировать на динамическом или статическом массиве *stack* и указателем на первый и последний добавленный элемент *head* и *tail*. Во втором случае, необходимо заранее знать максимальную возможную заполненность очереди. В нашем случае размер стека не может превосходить количества вершин графа.

Изначально все вершины считаются не посещенным. Добавляем первую вершину в очередь. Далее повторяем следующие действия: если очередь не пуста, вытаскиваем текущую вершину, и добавляем в очередь **всех** непосещенных соседей.

**Пример.** Начнем обход графа на рисунке 3 с вершины 1. Обход в глубину останется таким же. Рассмотрим обход в ширину. Рядом с каждым действием для наглядности написано состояние массивов *used* и *queue*.

| **Действие** | queue[7] | used[7] |
|---|---|---|
| | [1, 0, 0, 0, 0, 0, 0] | [1, 0, 0, 0, 0, 0, 0] |
| **Из очереди: вершина 1** | [0, 0, 0, 0, 0, 0, 0] | [1, 0, 0, 0, 0, 0, 0] |
| Ищем соседа 1: находим 3 (посещена). | [0, 3, 0, 0, 0, 0, 0] | [1, 0, 1, 0, 0, 0, 0] |
| Ищем соседа 1: находим 5 (посещена). | [0, 3, 5, 0, 0, 0, 0] | [1, 0, 1, 0, 1, 0, 0] |
| Ищем соседа 1: находим 7 (посещена). | [0, 3, 5, 7, 0, 0, 0] | [1, 0, 1, 0, 1, 0, 1] |
| Ищем соседа 1: нет. | [0, 3, 5, 7, 0, 0, 0] | [1, 0, 1, 0, 1, 0, 1] |
| **Из очереди: вершина 3.** | [0, 0, 5, 7, 0, 0, 0] | [1, 0, 1, 0, 1, 0, 1] |
| Ищем соседа 3: находим 2 (посещена). | [0, 0, 5, 7, 2, 0, 0] | [1, 1, 1, 0, 1, 0, 1] |
| Ищем соседа 3: находим 4 (посещена). | [0, 0, 5, 7, 2, 4, 0] | [1, 1, 1, 1, 1, 0, 1] |
| Ищем соседа 3: нет. | [0, 0, 5, 7, 2, 4, 0] | [1, 1, 1, 1, 1, 0, 1] |
| **Из очереди: вершина 5.** | [0, 0, 0, 7, 2, 4, 0] | [1, 1, 1, 1, 1, 0, 1] |
| Ищем соседа 5: нет. | [0, 0, 0, 7, 2, 4, 0] | [1, 1, 1, 1, 1, 0, 1] |
| **Из очереди: вершина 7.** | [0, 0, 0, 0, 2, 4, 0] | [1, 1, 1, 1, 1, 0, 1] |
| Ищем соседа 7: нет. | [0, 0, 0, 0, 2, 4, 0] | [1, 1, 1, 1, 1, 0, 1] |
| **Из очереди: вершина 2.** | [0, 0, 0, 0, 0, 4, 0] | [1, 1, 1, 1, 1, 0, 1] |
| Ищем соседа 2: нет. | [0, 0, 0, 0, 0, 4, 0] | [1, 1, 1, 1, 1, 0, 1] |
| **Из очереди: вершина 4.** | [0, 0, 0, 0, 0, 0, 0] | [1, 1, 1, 1, 1, 0, 1] |
| Ищем соседа 4: находим 6 (посещена). | [0, 0, 0, 0, 0, 0, 6] | [1, 1, 1, 1, 1, 1, 1] |
| Ищем соседа 4: нет. | [0, 0, 0, 0, 0, 0, 6] | [1, 1, 1, 1, 1, 1, 1] |
| **Из очереди: вершина 6.** | [0, 0, 0, 0, 0, 0, 0] | [1, 1, 1, 1, 1, 1, 1] |
| Ищем соседа 6: нет. | [0, 0, 0, 0, 0, 0, 0] | [1, 1, 1, 1, 1, 1, 1] |

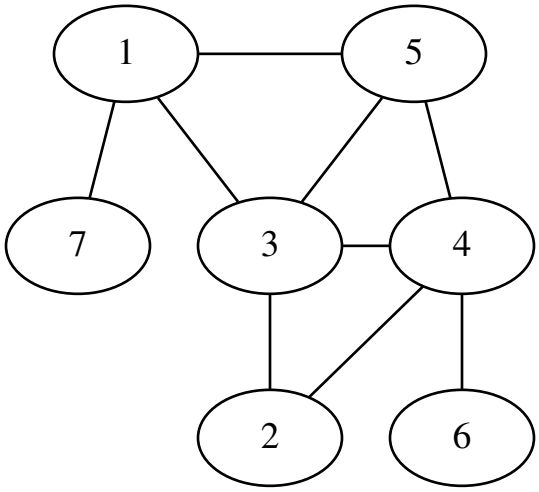


Fig. 3: Пример графа.

**Псевдокод.** Добавление в стек на статическом массиве (в конец). Указатель *tail* указывает на позицию за последним элементом.

```
stackPush(value)
    stack[tail] := 1
    tail := tail + 1
```

Добавление в стек на статическом массиве (в конец). Указатель *tail* указывает на позицию за последним элементом.

```
stackPush(value)
    stack[tail] := value
    tail := tail + 1
```

Извлечение из стека на статическом массиве (из конца). Указатель *tail* указывает на позицию за последним элементом.

```
stackPop()
    tail := tail - 1
    return stack[tail]
```

Добавление в очередь на статическом массиве (в конец). Указатель *tail* указывает на позицию за последним элементом.

```
queuePush(value)
    queue[tail] := value
    tail := tail + 1
```

Извлечение из очереди на статическом массиве (из конца). Указатель *head* указывает на позицию первого элемента очереди.

```
queuePop()
    head := head + 1
    return queue[head - 1]
```

Обход в глубину для матрицы смежности `incendMatrix`.

```
dfs(start)
    stackPush(start)
    used[start] := 1
    print(start)
```

```
    while stackSize() > 0 do
        current := stackPop()
        for next := 1 .. n do
            if (incendMatrix[current][next] = 1) and (used[next] = 0) then
                stackPush(next)
                used[next] := 1
                print(next)
                break
```

Обход в ширину для матрицы смежности `incendMatrix`.

```
bfs(start)
    queuePush(start)
    used[start] := 1
    print(start)

    while stackSize() > 0 do
        current := queuePop()
        for next := 1 .. n do
            if (incendMatrix[current][next] = 1) and (used[next] = 0) then
                queuePush(next)
                used[next] := 1
                print(next)
```

**Асимптотика.** Удобно считать за действие количество операций сравнений `used[next] = 0`. В худшем случае при обходе в глубину с матрицей смежности будут посещены все $n$ вершин и из каждой будут сделаны $n$ таких проверок. Итог: $\Theta\left(n^3\right)$. В худшем случае при обходе в глубину со списком смежных ребер будут сделаны переходы по всем ребрам. Итог: $\Theta\left(m\right)$.

**Практика.** Даны 2 целых числа: $n$ — количество вершин графа и $m$ — количество ребер. Далее $m$ пар чисел $v_i$ и $w_i$ — номера вершин, соответствующие $i$-му ребру. Вывести порядок вершин при обходе данного графа в глубину. Сравнить обход в глубину и в ширину при матрице смежности.

## 4 Поиск пути в лабиринте

Дан лабиринт в виде таблицы, которая описывает лабиринт (каждая клетка может быть либо свободной, либо нет). Необходимо найти путь между данными 2 клетками таблицы.

**Метод решения.** Вершины графа — это свободные клетки (определяется 2 параметрами: номер строки и номер столбца). Ребра — переходы между соседними по стороне клетками (определяется одним из 4 направлений: вниз, вправо, влево, вверх). Такая постановка позволяет избавить от дополнительных структур типа матрицы смежности или списка смежных ребер, так как переходы их можно просчитать во время выполенения обхода. Для поиска пути будем применять рекурсивный обход в глубину или нерекурсивный обход в ширину.

Для отметок о посещенных вершин используется матрицу `used`, причем будем отмечать направление, из которого попадаем в данную клетку (число от 1 до 4). Для переходов по соседям — массивы переходов:

```
di[4] = {1, 0, 0, -1}
dj[4] = {0, 1, -1, 0}
```

Для нерекурсивного обхода понадобиться в очередь укладывать и извлекать по 2 элемента (номер строки и номер столбца).

**Пример.** Лабиринт задан в виде таблицы символов: 'X' — несвободная клетка, '.' — свободная клетка, 'S' — начало пути, 'F' — конец пути.

| | | | | |
|---|---|---|---|---|
| X | S | X | . | X |
| X | . | X | . | F |
| . | . | X | . | X |
| X | . | . | . | . |