ГРАФЫ: часть 1

Лекторы
Пермяков Антон Сергеевич
Ткаченко Данил Михайлович

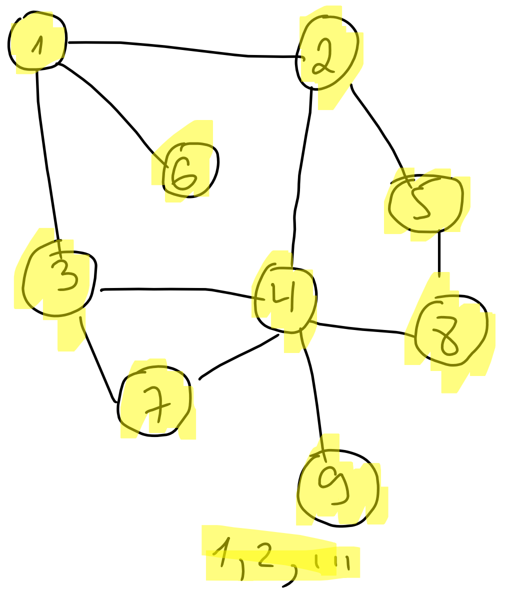
АЛГОРИТМЫ

1. Обход в ширину
   - Подсчет длины пути
2. Обход в глубину
   - Поиск цикла
3. Топологическая сортировка
4. Поиск компонент связности (+ слабой связности)
5. Поиск компонент сильной связности
   - Конденсация графа (ориентированного)

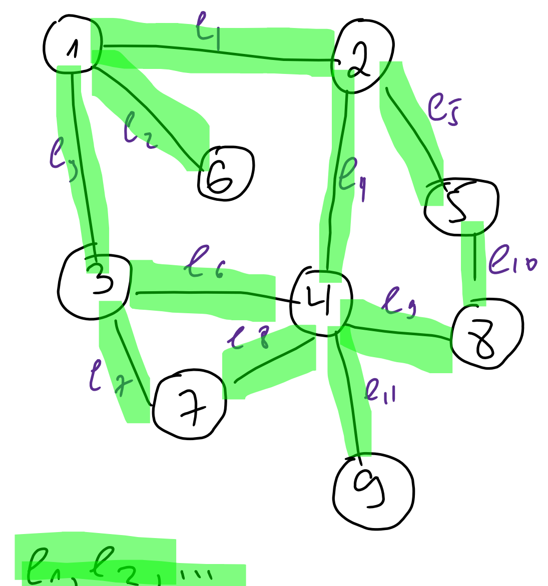
Обходы графов?

Обходы графов

По вершинам $\{V\}$

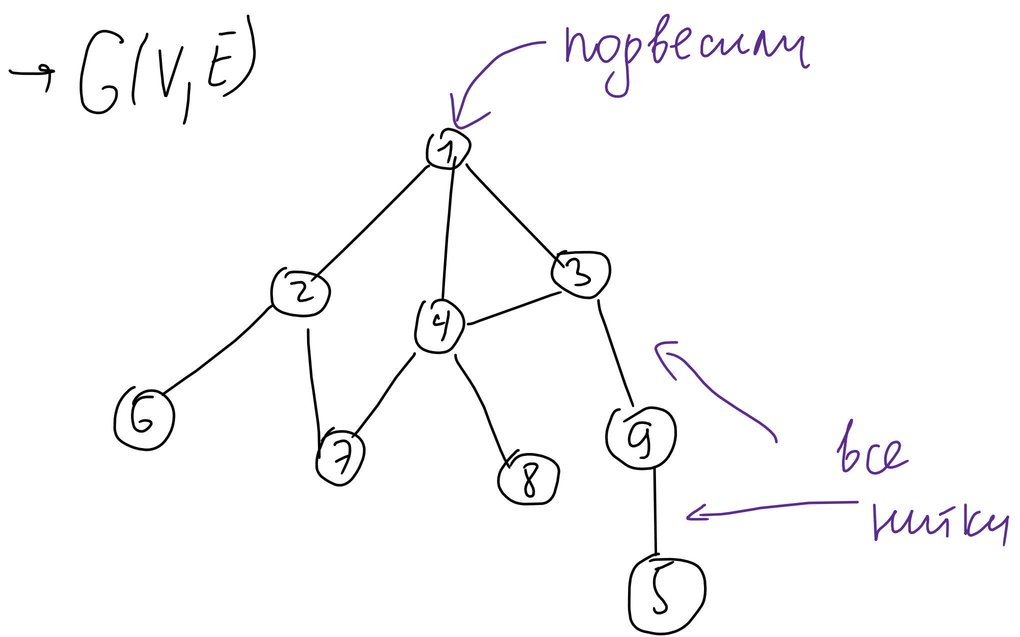


По ребрам/дугам $\{E\}$



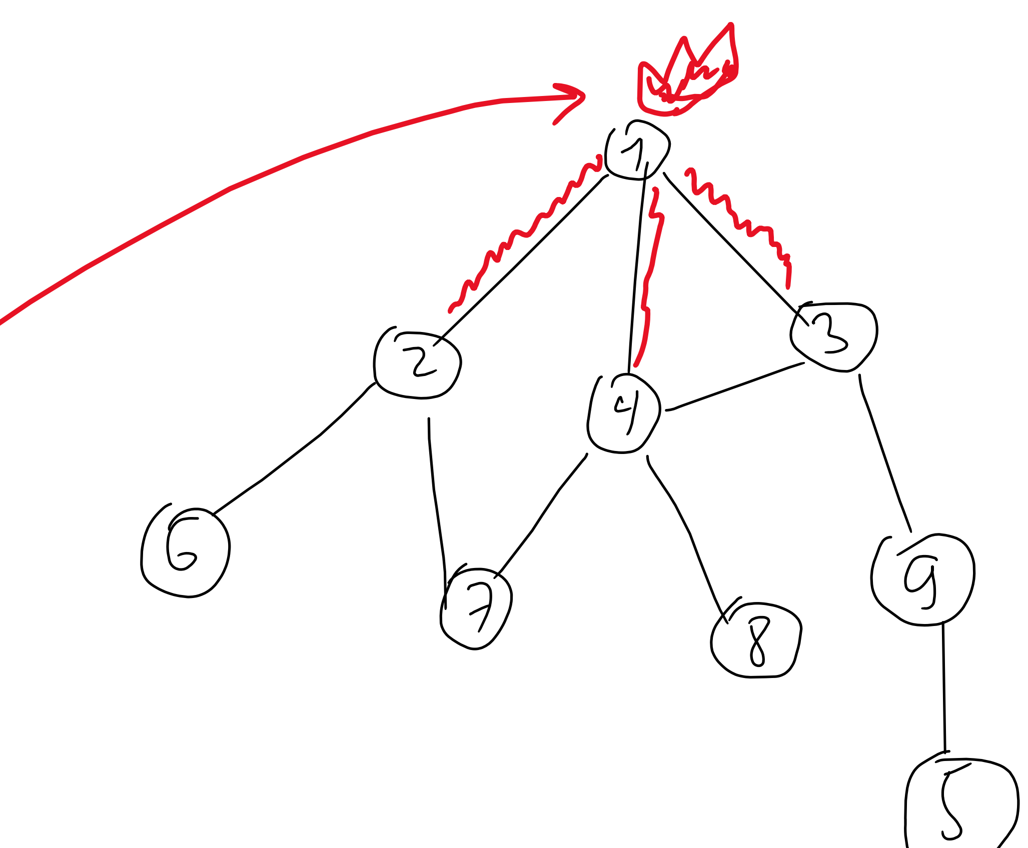
Обход в ширину

- Пусть РЕБРА - нитки.
- Подвесим наш граф за вершину **1**



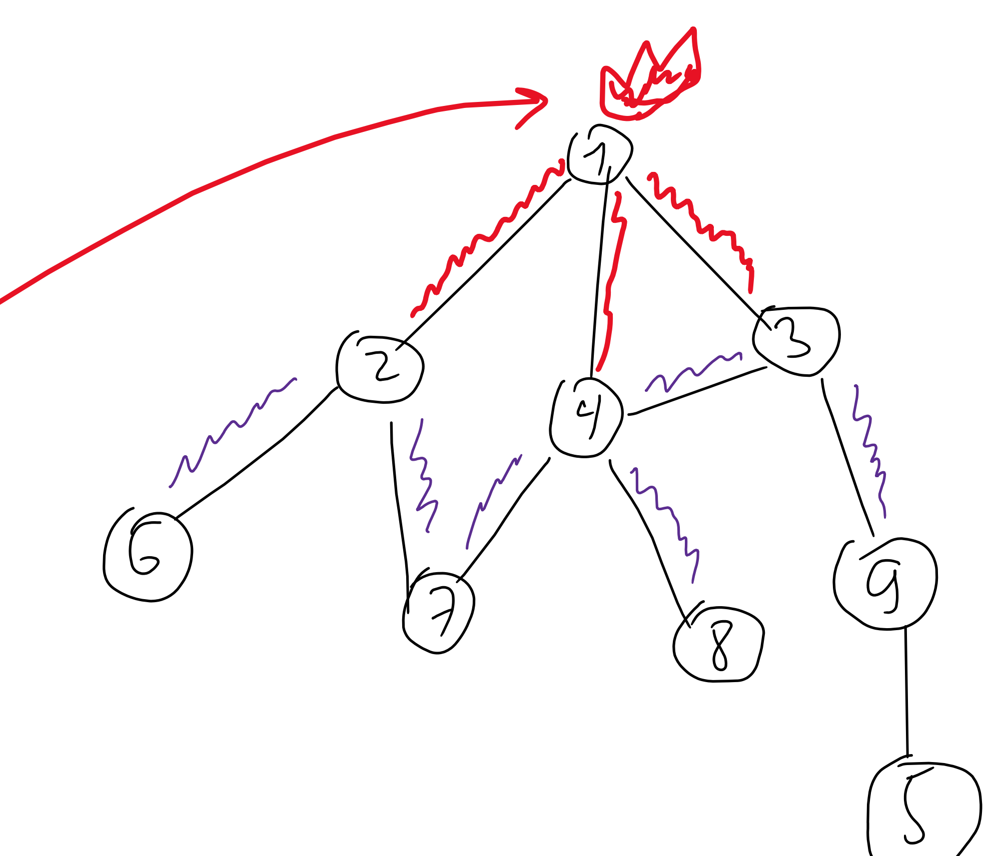
Обход в ширину

- Пусть РЕБРА - нитки.
- Подвесим наш граф за вершину **1**
- **Подожжём нитки из вершины 1**



Обход в ширину

- Пусть РЕБРА - нитки.
- Подвесим наш граф за вершину **1**
- **Подожжём нитки из вершины 1**

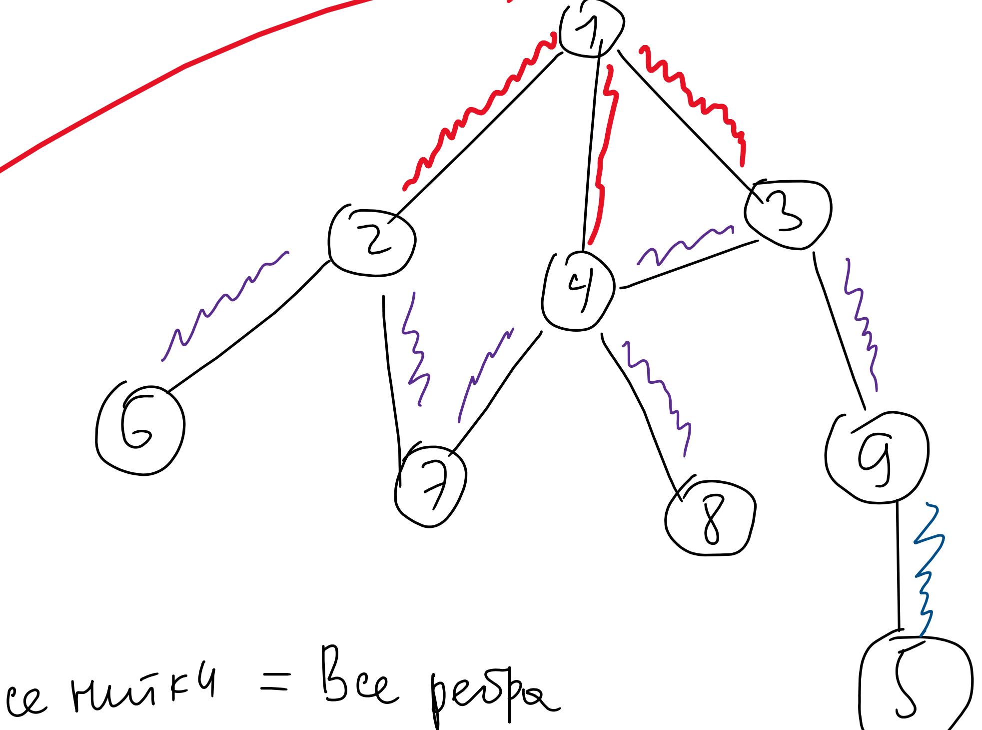


Обход в ширину

- Пусть РЕБРА - нитки.
- Подвесим наш граф за вершину **1**
- **Подожжём нитки из вершины 1**
- Все сгорело за 3 этапа

Обход в ширину

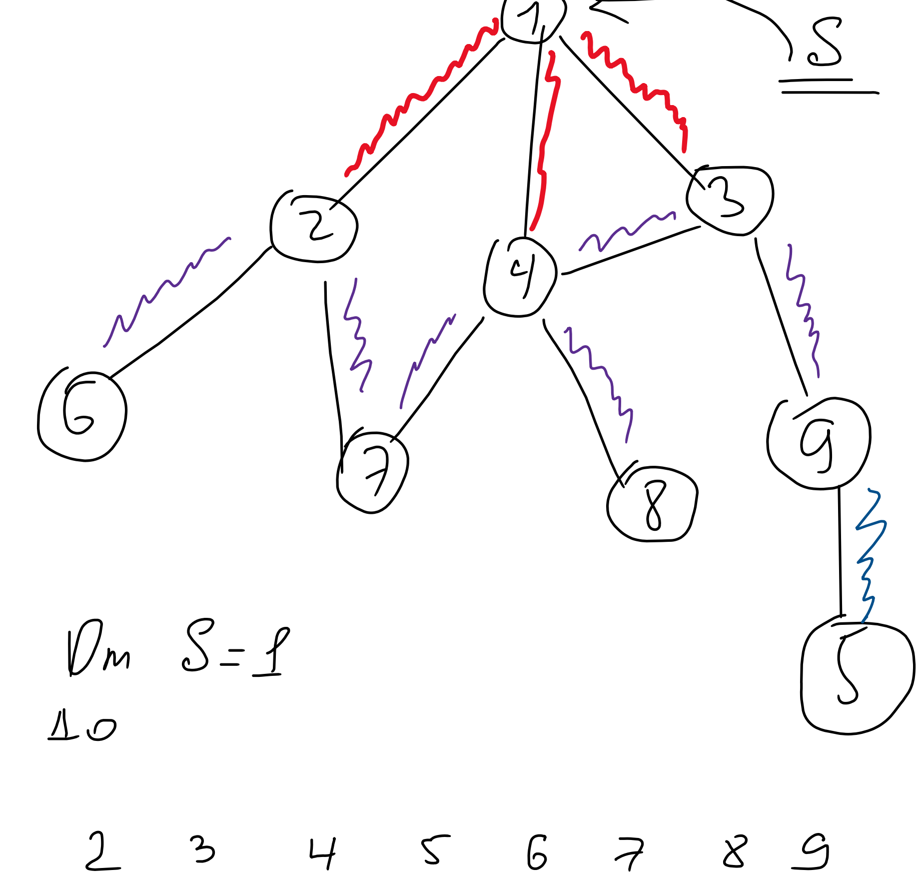
- Пусть РЕБРА - нитки.
- Подвесим наш граф за вершину **1**
- **Подожжём нитки из вершины 1**
- Все сгорело за 3 этапа



это что? = Все нитки = Все ребра

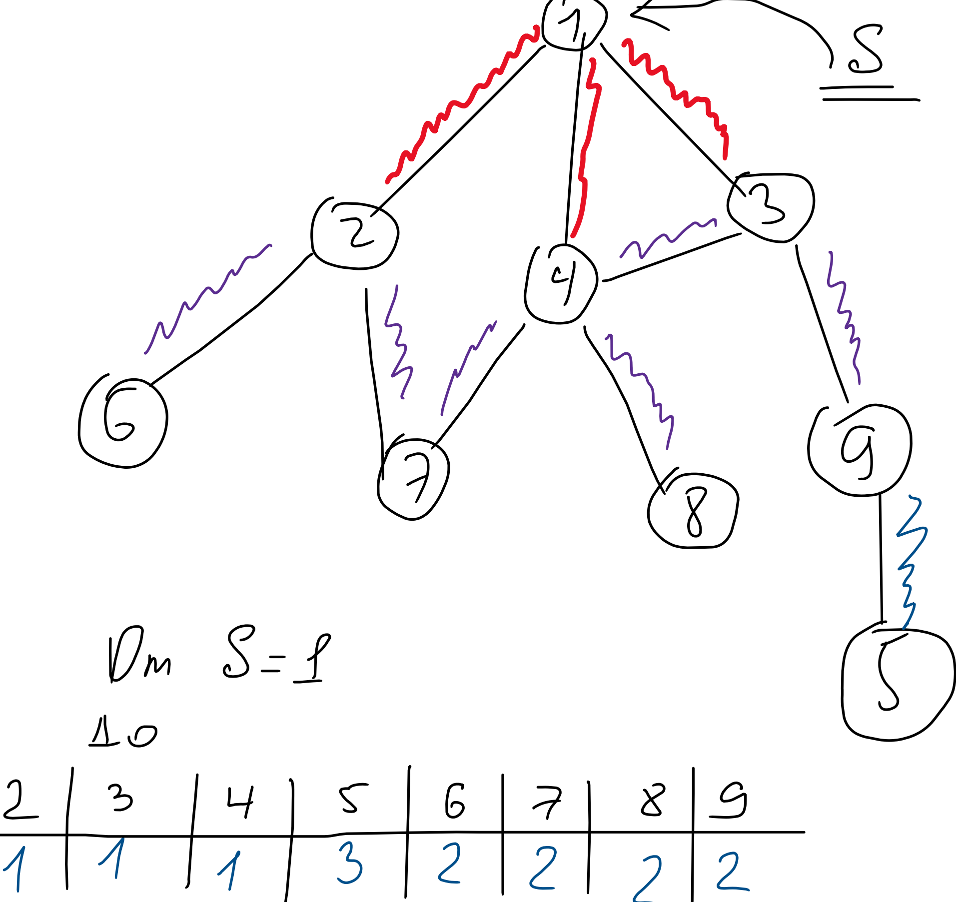
Обход в ширину

- Пусть РЕБРА - нитки.
- Подвесим наш граф за вершину **1**
- **Подожжём нитки из вершины 1**
- Все сгорело за 3 этапа



Обход в ширину

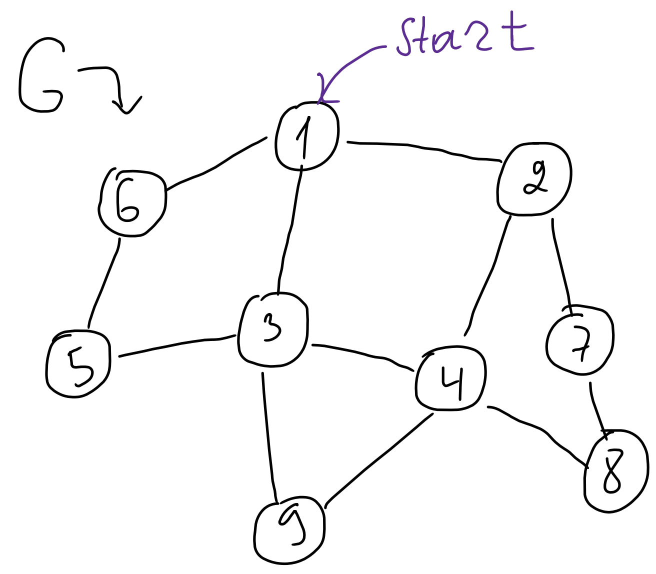
- Пусть РЕБРА - нитки.
- Подвесим наш граф за вершину **1**
- **Подожжём нитки из вершины 1**
- Все сгорело за 3 этапа



Обход в ширину

- Обход по ребрам/дугам
- Алгоритм поиска в ширину в невзвешенном графе находит **длины кратчайших путей** до всех достижимых вершин от заданной.

Обход в ширину

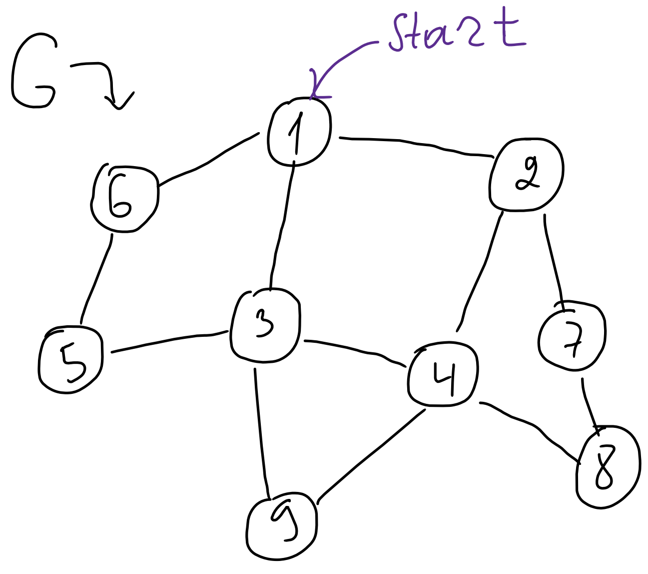


Как будет работать алгоритм ?

Когда закончится алгоритм обхода?

Как хранить граф ?

Обход в ширину



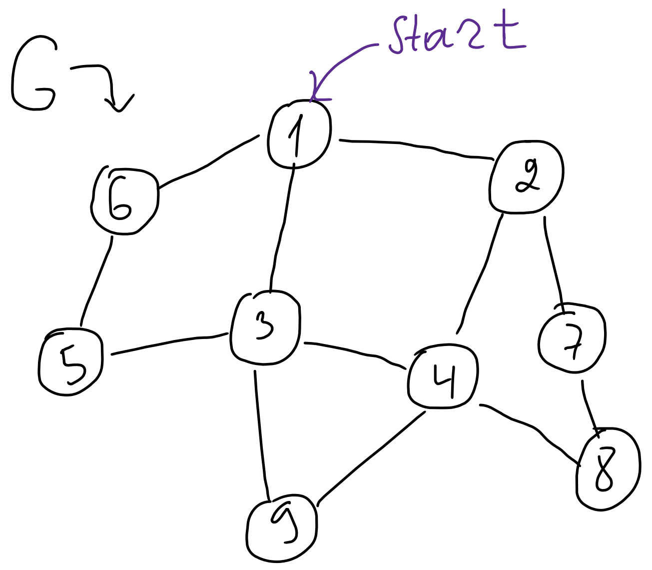
Как будет работать алгоритм ?

Горят все соседние
⇒ идём по соседям от сгоревших

Когда закончится алгоритм обхода?

Как хранить граф ?

Обход в ширину



Как будет работать алгоритм ?

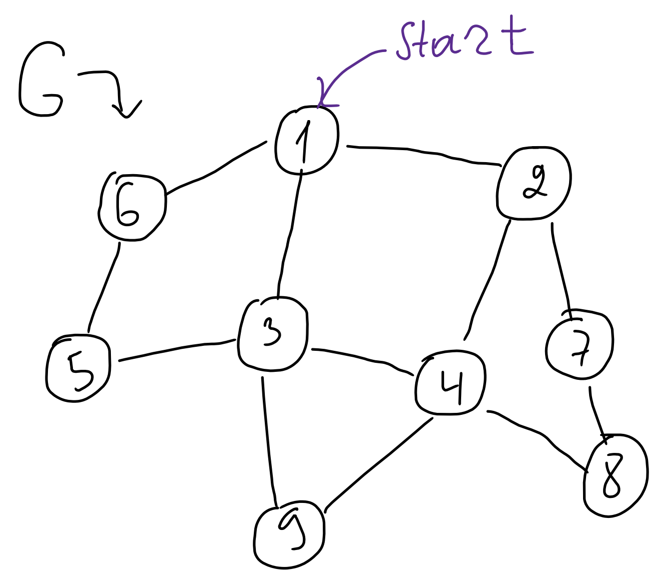
Горят все соседние
⇒ идём по соседям от сгоревших

Когда закончится алгоритм обхода?

все сгорело ⇒ помечаем сгоревшие !

Как хранить граф ?

Обход в ширину



Как будет работать алгоритм ?

Горят все соседние
=> идём по соседям от сгоревших

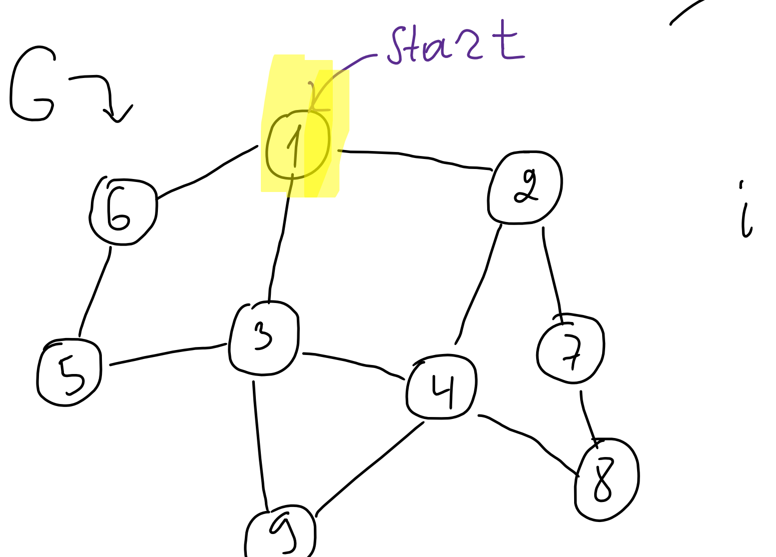
Когда закончится алгоритм обхода?

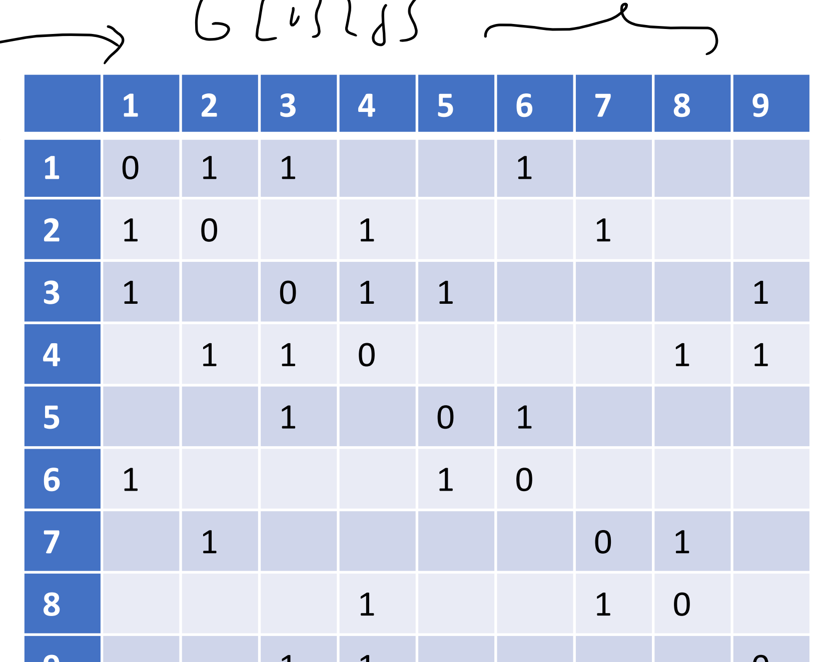
все сгорело => помечаем сгоревшие!

Как хранить граф ?

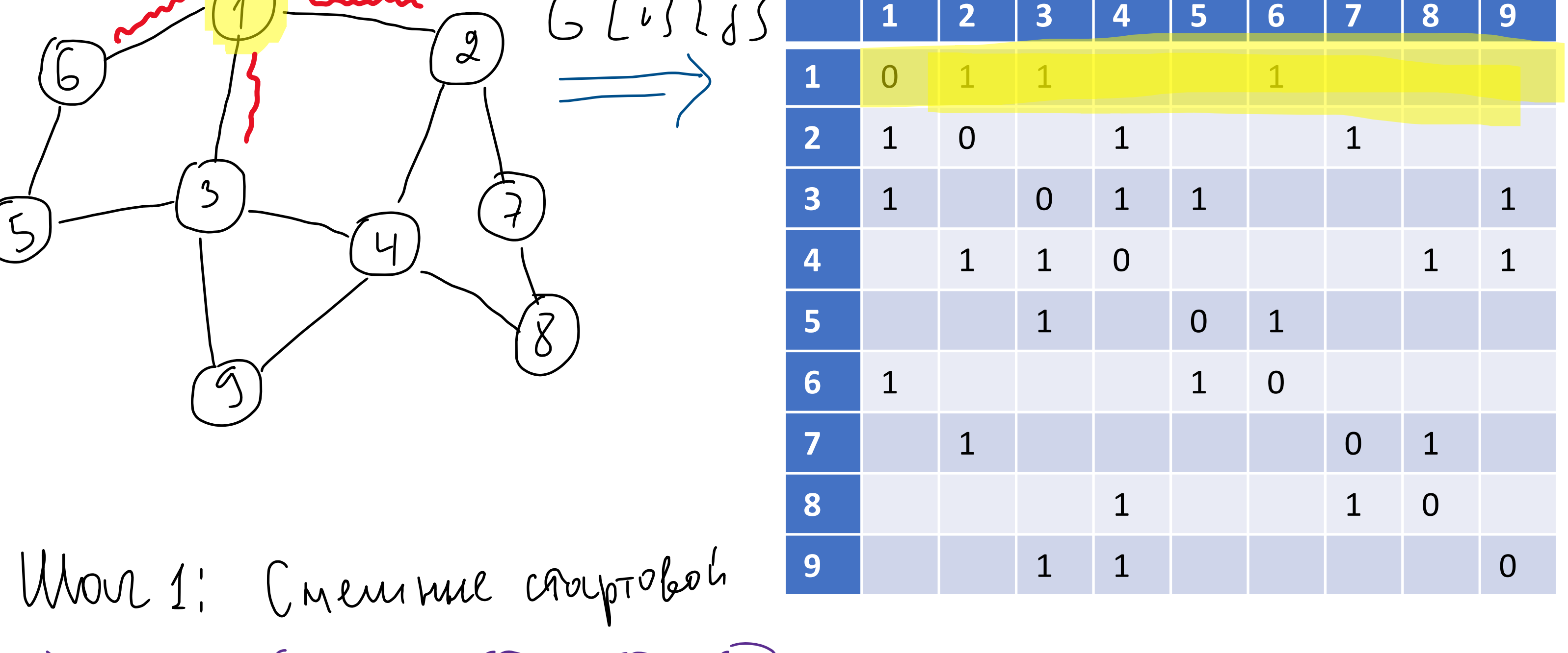
Соседи => смежность вершин =>
матрица смежности

Обход в ширину



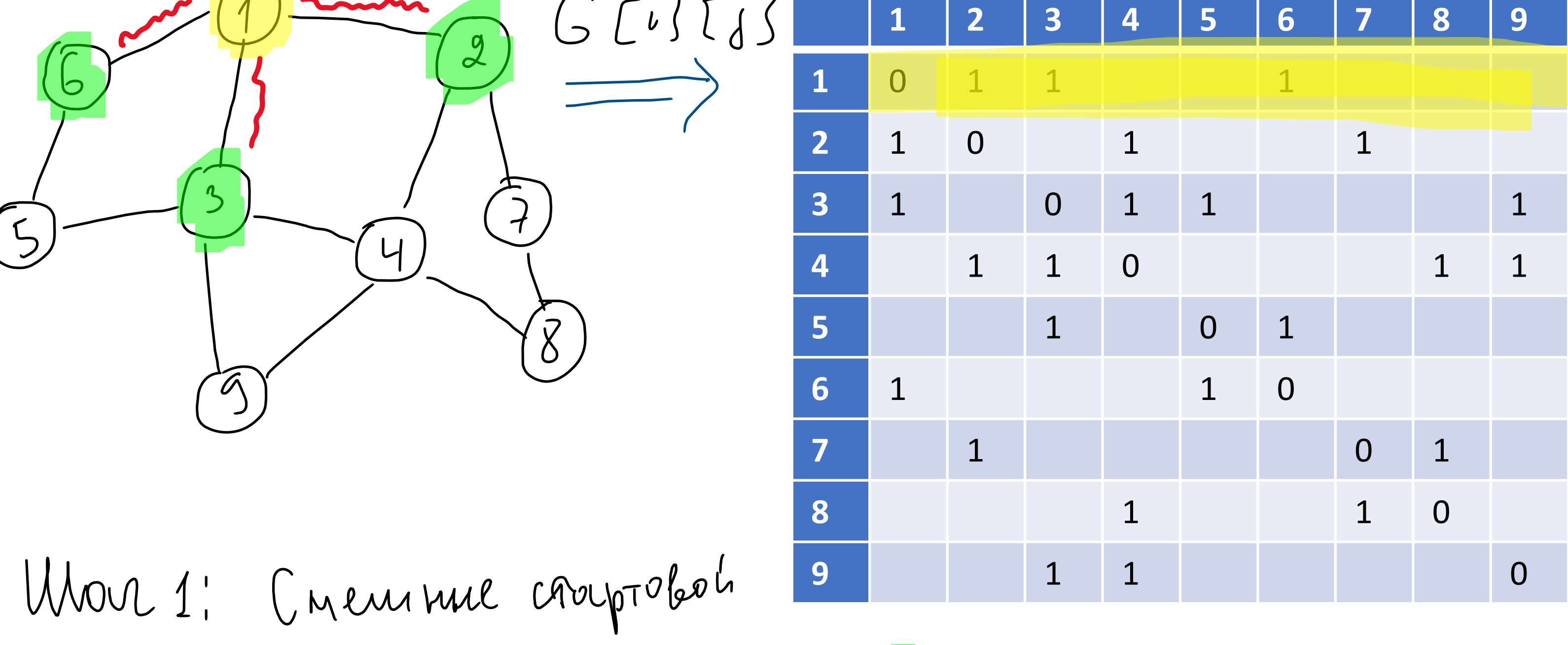


	1	2	3	4	5	6	7	8	9
1	0	1	1			1			
2	1	0		1			1		
3	1		0	1	1				1
4		1	1	0				1	1
5			1		0	1			
6	1				1	0			
7		1					0	1	
8				1			1	0	
9			1	1					0



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

⇒ для ① это ⑥, ③, ②

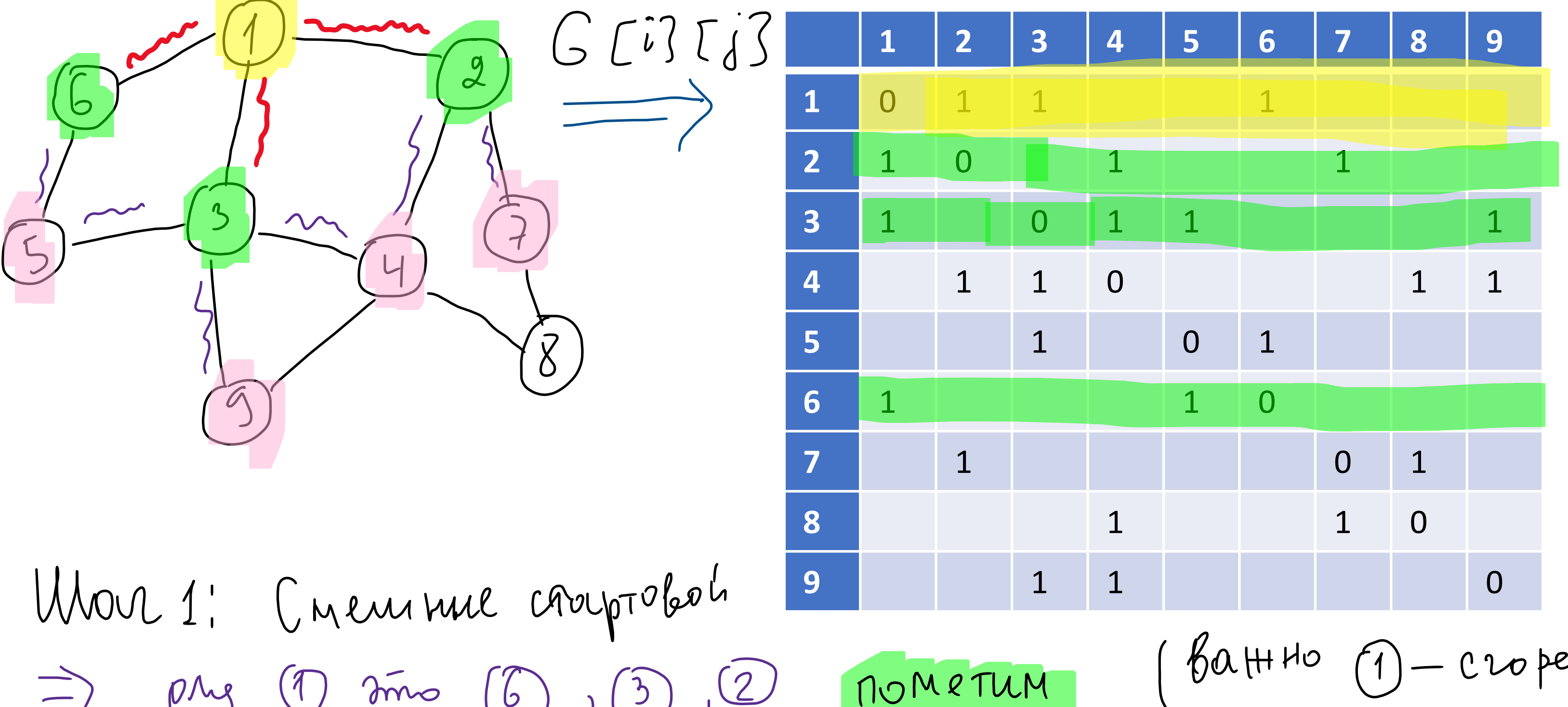


	1	2	3	4	5	6	7	8	9
1	0	1	1			1			
2	1	0		1			1		
3	1		0	1	1				1
4		1	1	0				1	1
5			1		0	1			
6	1				1	0			
7		1					0	1	
8				1			1	0	
9			1	1					0

Шаг 1: Смежные стартовой

⇒ для ① это ⑥, ③, ② пометим

Шаг 2:



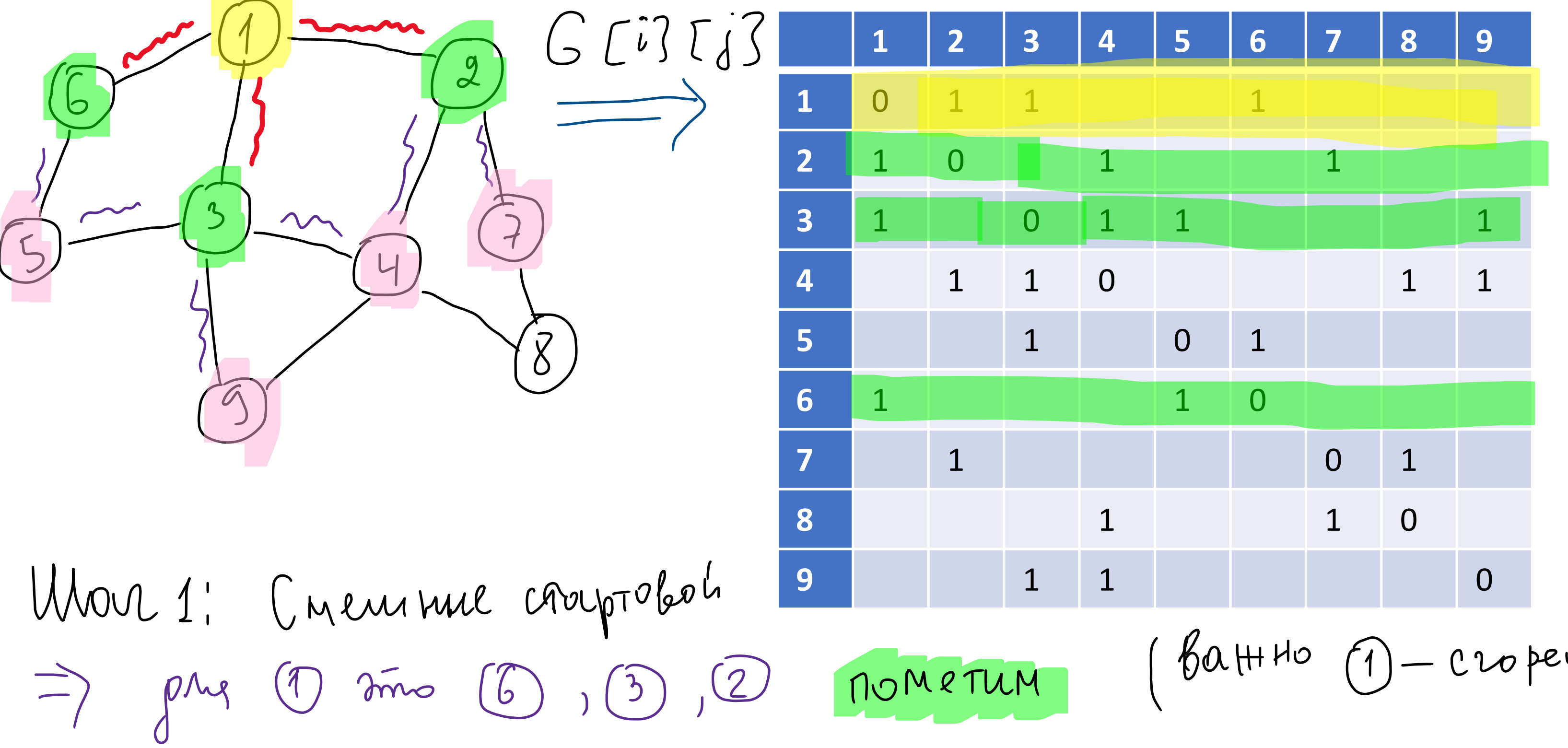
	1	2	3	4	5	6	7	8	9
1	0	1	1			1			
2	1	0		1			1		
3	1		0	1	1				1
4		1	1	0				1	1
5			1		0	1			
6	1				1	0			
7		1					0	1	
8				1			1	0	
9			1	1					0

Шаг 1: Смежные стартовой

⇒ для ① это ⑥, ③, ② **пометим** (важно ① — сгорела!)

Шаг 2: рассмотрим ⑥ ③ ② → смежные им

⇒



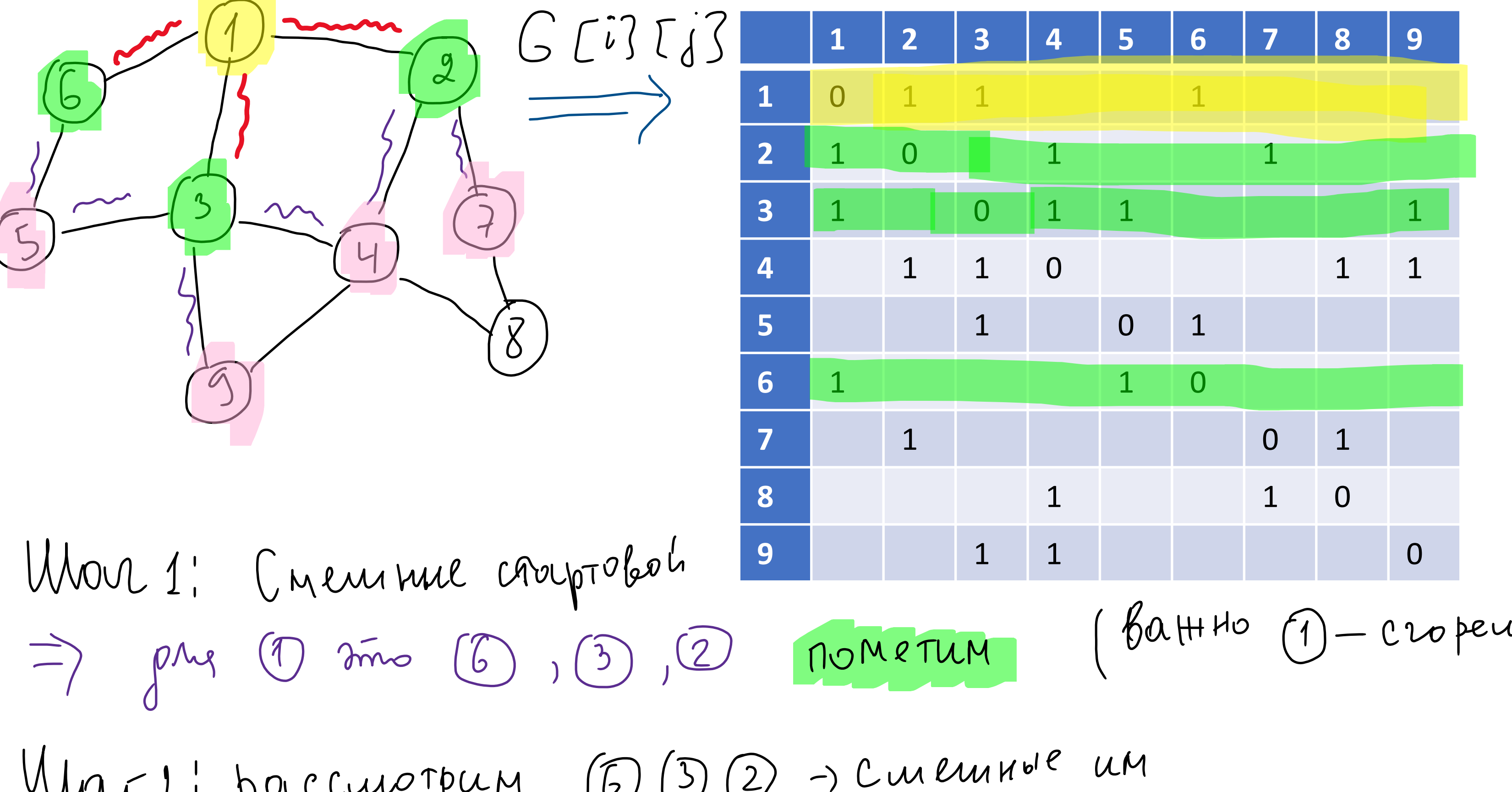
	1	2	3	4	5	6	7	8	9
1	0	1	1			1			
2	1	0		1			1		
3	1		0	1	1				1
4		1	1	0				1	1
5			1		0	1			
6	1				1	0			
7		1					0	1	
8				1			1	0	
9			1	1					0

Шаг 1: Смежные стартовой

⇒ для (1) это (6), (3), (2) **пометим** (важно (1) — сгорела!)

Шаг 2: рассмотрим (6) (3) (2) → смежные им

⇒ это будет (5) (9) (4) (7)



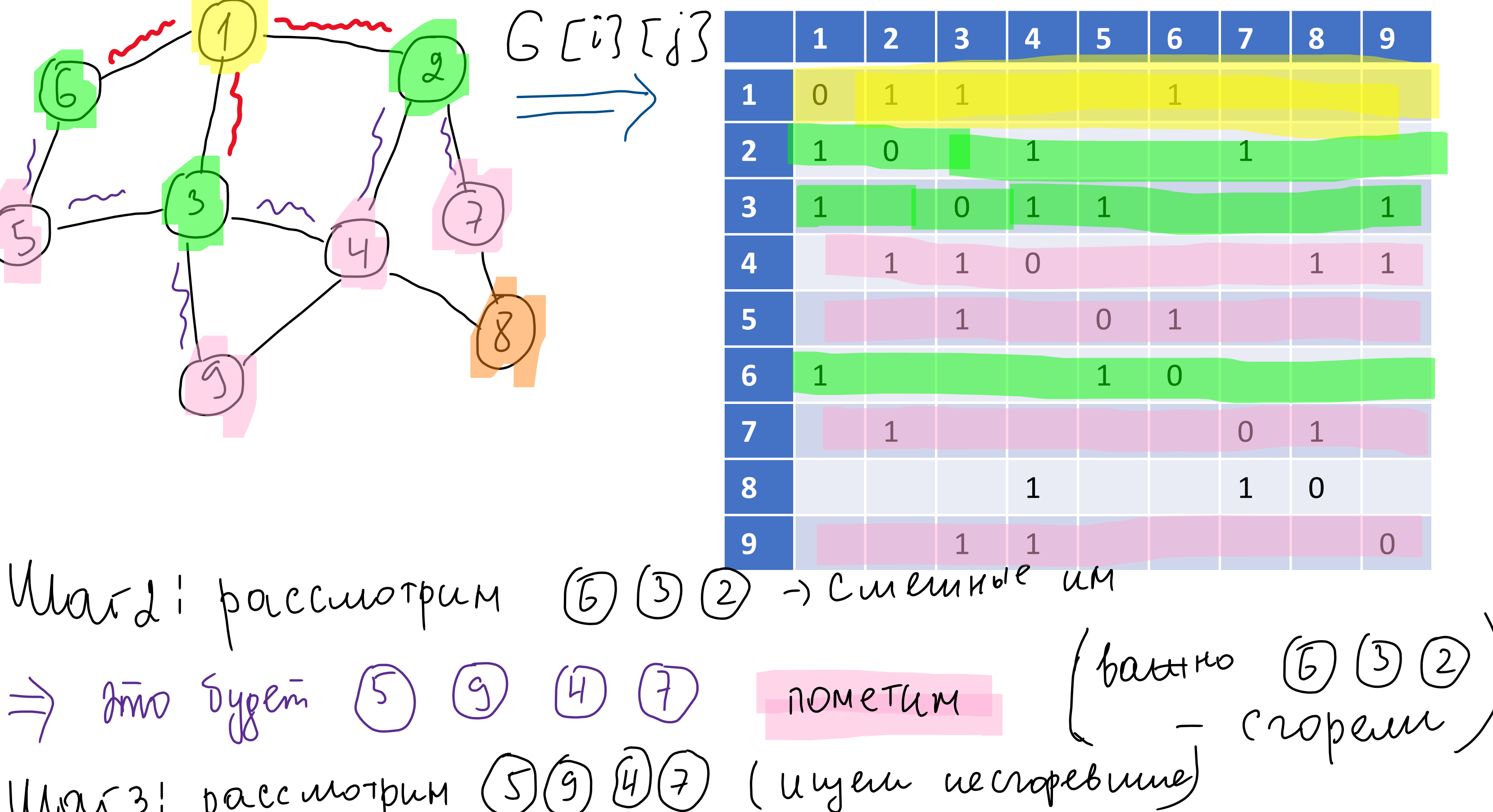
	1	2	3	4	5	6	7	8	9
1	0	1	1			1			
2	1	0		1			1		
3	1		0	1	1				1
4		1	1	0				1	1
5			1		0	1			
6	1				1	0			
7		1					0	1	
8				1			1	0	
9			1	1					0

Шаг 1: Смежные стартовой

⇒ для (1) это (6), (3), (2) пометим (важно (1) — сгорела!)

Шаг 2: рассмотрим (6) (3) (2) → смежные им

⇒ это будет (5) (9) (4) (7) пометим (важно (6) (3) (2) — сгорели)



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

Шаг 2: рассмотрим (6) (3) (2) → смежные им

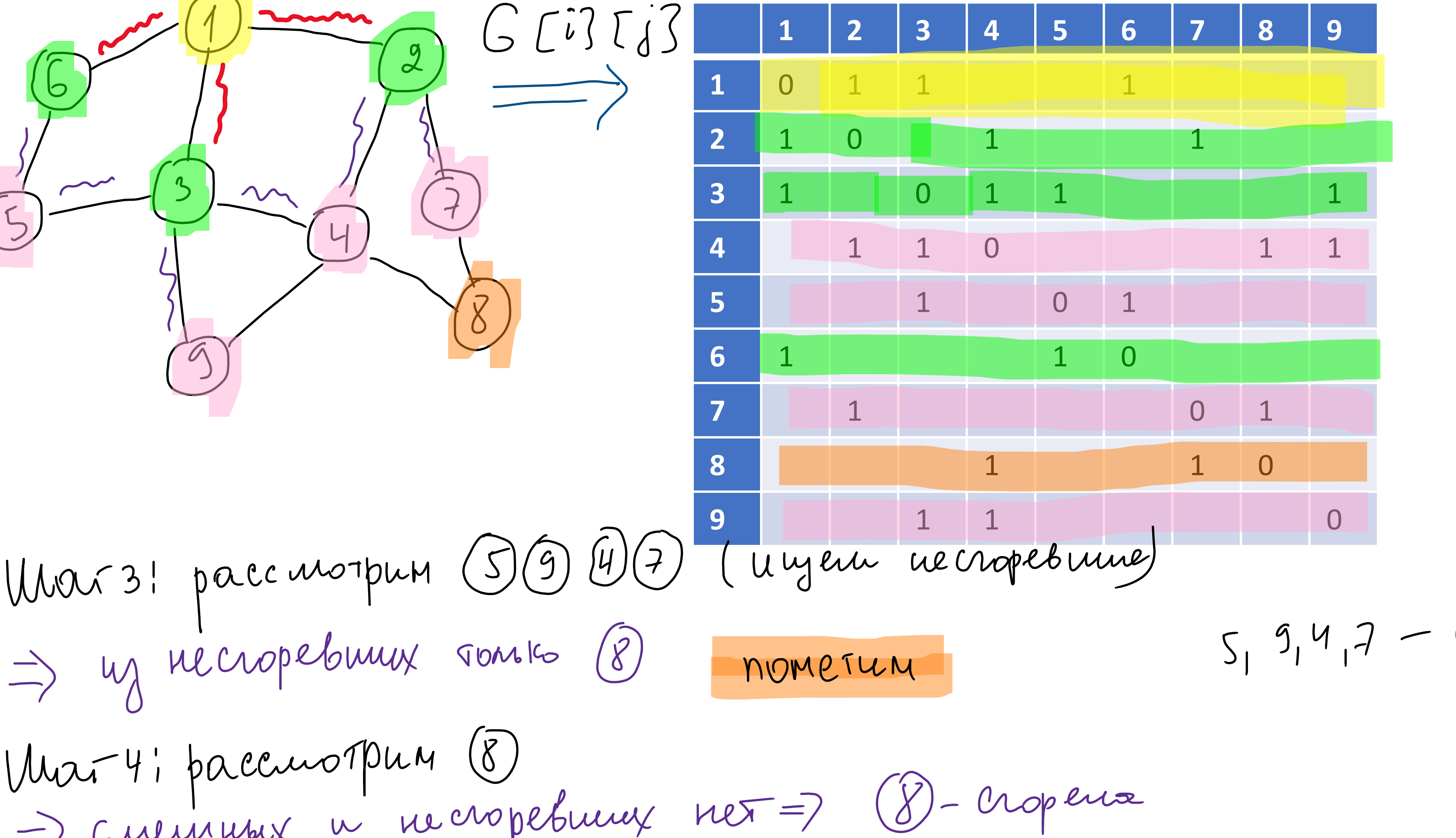
⇒ это будет (5) (9) (4) (7) пометим

(важно (6) (3) (2) – сгорели)

Шаг 3: рассмотрим (5) (9) (4) (7) (ищем несгоревшие)

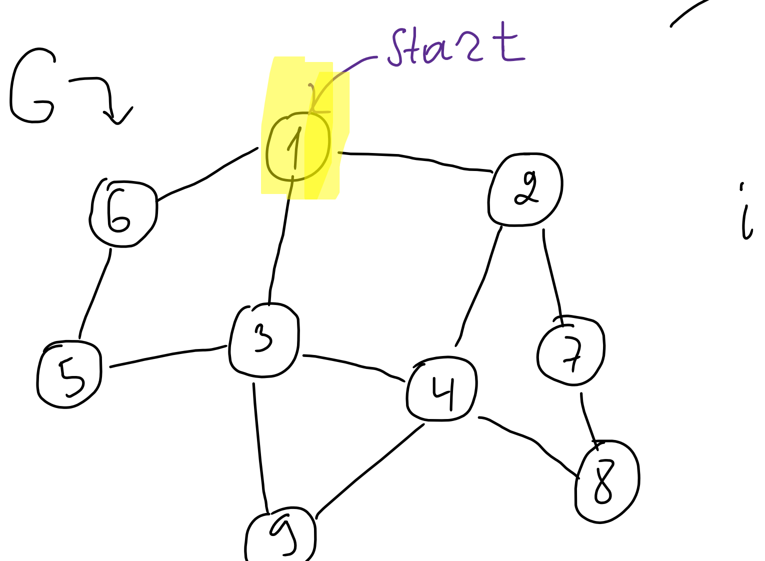
⇒ из несгоревших только (8) пометим

5, 9, 4, 7 – сгорели



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

Обход в ширину



G[i][j]

j

i

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

! j = идёт по всем вершинам 1..9 | i проходит также все вершины

из которой i, что горит

+ две метки сгорели, горит

Обход в ширину

- Какие идеи эффективной реализации?

- Как найти расстояние до всех вершин от стартовой ?

- Можно ли хранить граф иначе для обхода в ширину?

Обход в ширину

- Какие идеи эффективной реализации?
  - *Используем очередь для отслеживания вершин, что горят*
  - *Убрали из очереди – пометили, что сгорела*
  - *Рассматриваем в один момент времени – одну вершину*
- Как найти расстояние до всех вершин от стартовой ?
  - *Использовать дополнительный массив для подсчета расстояния от стартовой до всех остальных*
  - *+ Массив предков для восстановления путей*
- Можно ли хранить граф иначе для обхода в ширину?
  - *Список смежности/Матрица инцидентности*

Обходы в ширину:
BFS (G(V, E) , s – стартовая вершина)

!!!

- 1. Помечаем все вершины как не пройденные
- 2. Добавляем в очередь стартовую вершину
- 3. В цикле: (пока очередь не пустая)
  - a. Берем вершину из очереди и помечаем ее как пройдённую
  - b. Добавляем в очередь все смежные с ней вершины, которые не пройденные
  - c. Удаляем из очереди пройденную вершину

по одной вершине рассматр

В ширину: BFS

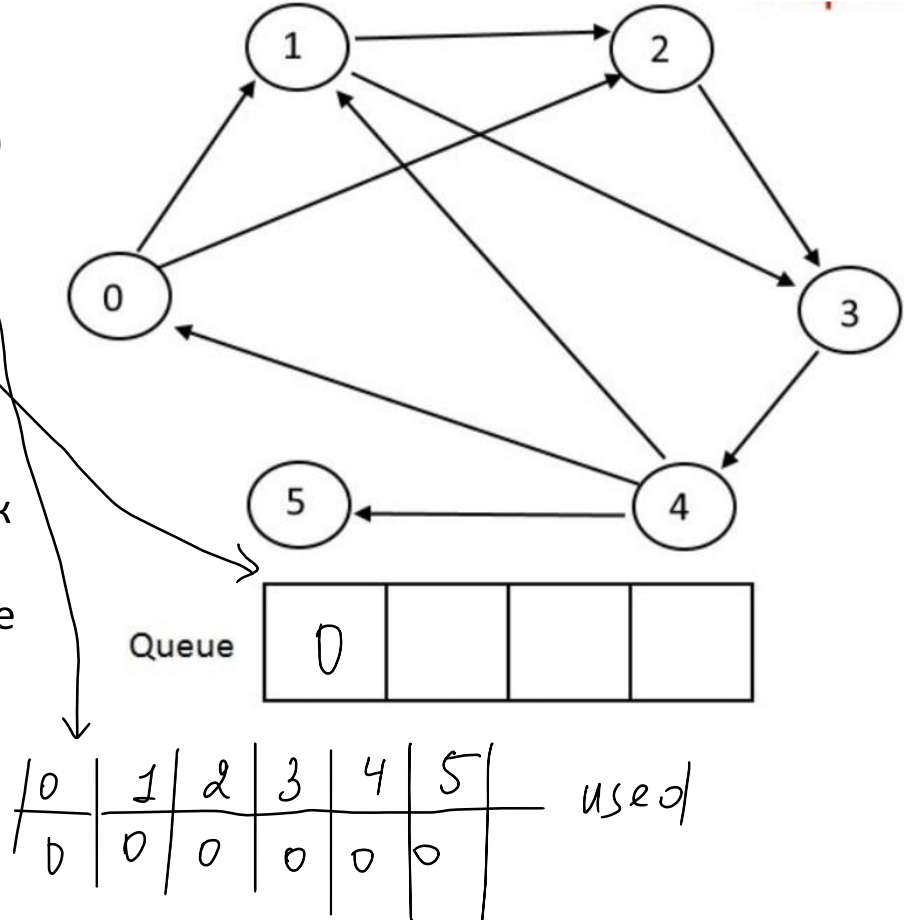
- 1. Помечаем все вершины как не пройденные
- 2. Добавляем в очередь стартовую вершину
- 3. В цикле: (пока очередь не пустая)
  - a. Берем вершину из очереди и помечаем ее как пройдённую
  - b. Добавляем в очередь все смежные с ней вершины, которые не пройденные
  - c. Удаляем из очереди пройденную вершину

Step 1

Queue

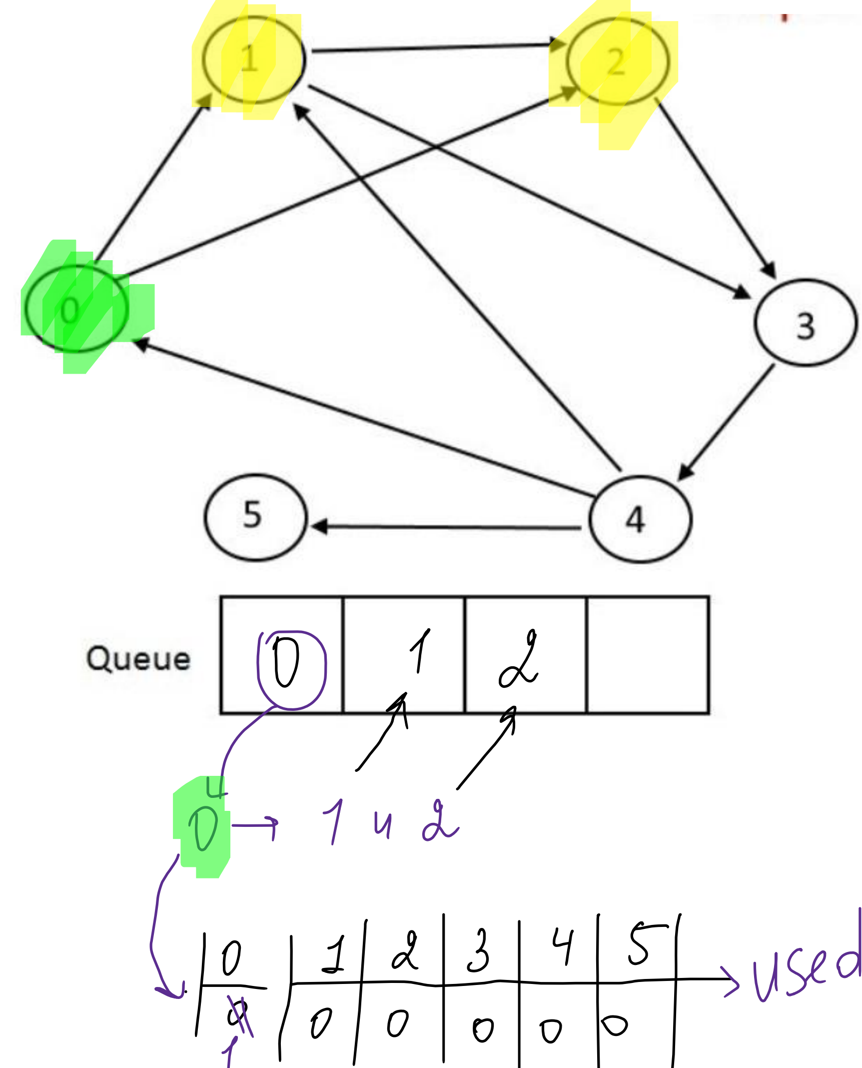
В ширину: BFS

- 1. Помечаем все вершины как не пройденные
- 2. Добавляем в очередь стартовую вершину
- 3. В цикле: (пока очередь не пустая)
  - a. Берем вершину из очереди и помечаем ее как пройдённую
  - b. Добавляем в очередь все смежные с ней вершины, которые не пройденные
  - c. Удаляем из очереди пройденную вершину



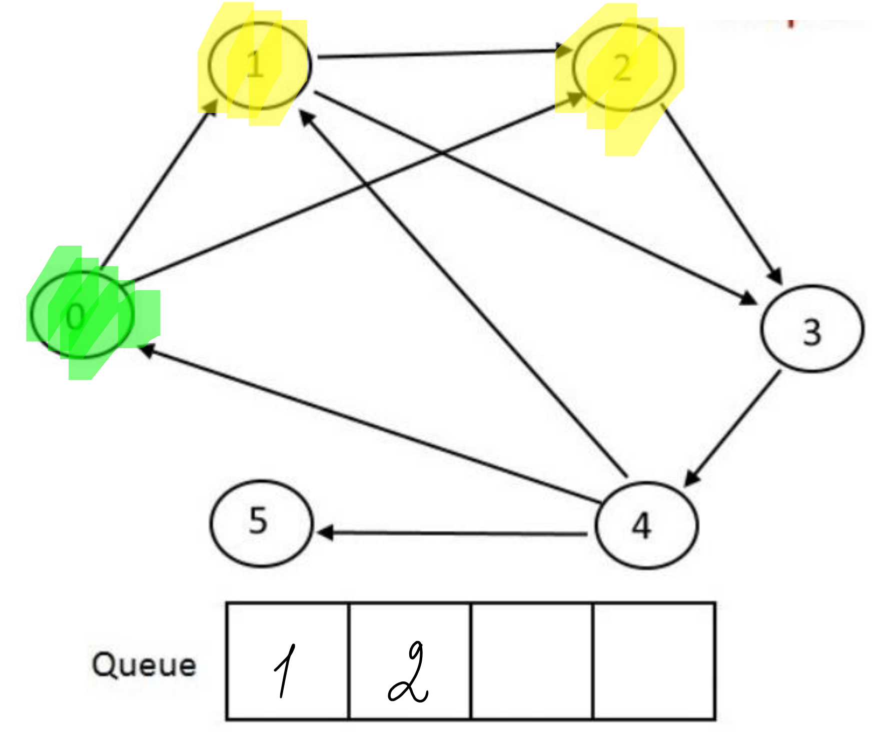
В ширину: BFS

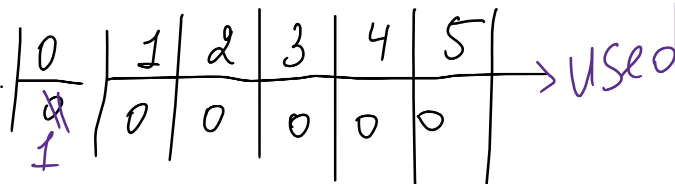
- 1. Помечаем все вершины как не пройденные
- 2. Добавляем в очередь стартовую вершину
- 3. В цикле: (пока очередь не пустая)
  - a. Берем вершину из очереди и помечаем ее как пройдённую
  - b. Добавляем в очередь все смежные с ней вершины, которые не пройденные
  - c. Удаляем из очереди пройденную вершину



В ширину: BFS

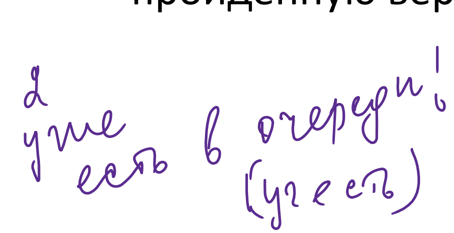
- 1. Помечаем все вершины как не пройденные
- 2. Добавляем в очередь стартовую вершину
- 3. В цикле: (пока очередь не пустая)
  - a. Берем вершину из очереди и помечаем ее как пройдённую
  - b. Добавляем в очередь все смежные с ней вершины, которые не пройденные
  - c. Удаляем из очереди пройденную вершину





В ширину: BFS

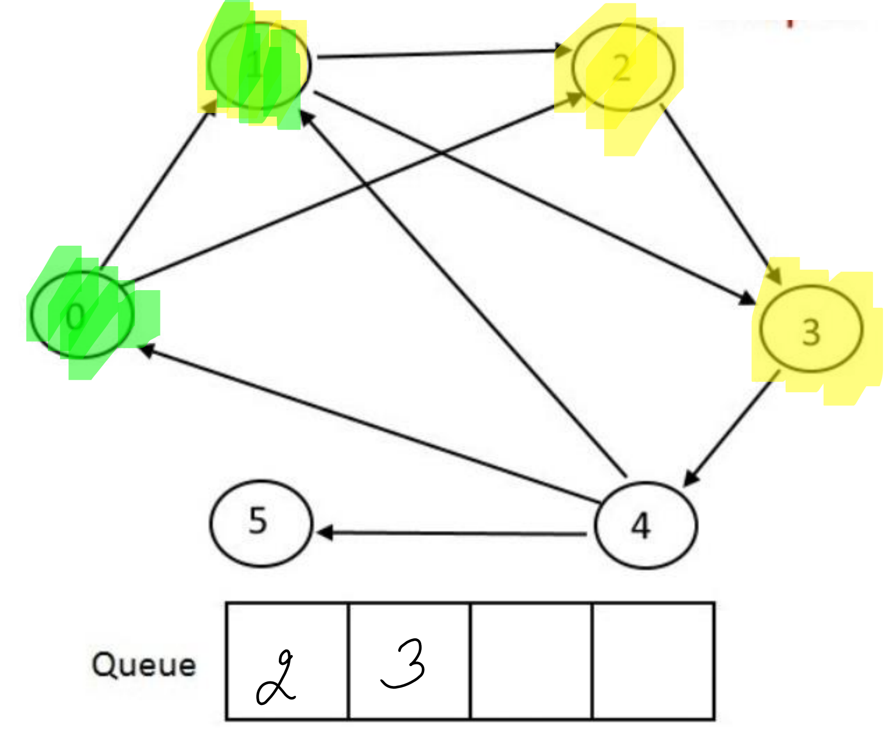
- 1. Помечаем все вершины как не пройденные
- 2. Добавляем в очередь стартовую вершину
- 3. В цикле: (пока очередь не пустая)
  - a. Берем вершину из очереди и помечаем ее как пройдённую
  - b. Добавляем в очередь все смежные с ней вершины, которые не пройденные
  - c. Удаляем из очереди пройденную вершину



Queue

В ширину: BFS

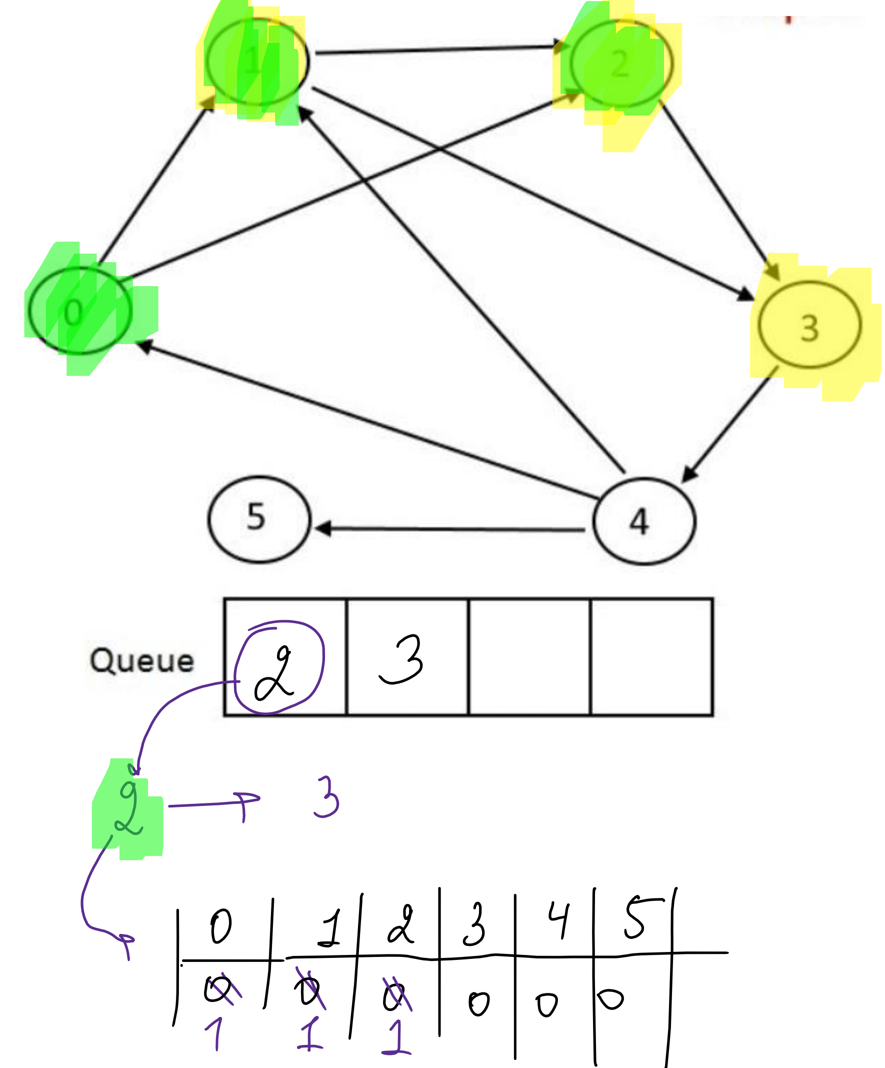
- 1. Помечаем все вершины как не пройденные
- 2. Добавляем в очередь стартовую вершину
- 3. В цикле: (пока очередь не пустая)
  - a. Берем вершину из очереди и помечаем ее как пройдённую
  - b. Добавляем в очередь все смежные с ней вершины, которые не пройденные
  - c. Удаляем из очереди пройденную вершину



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

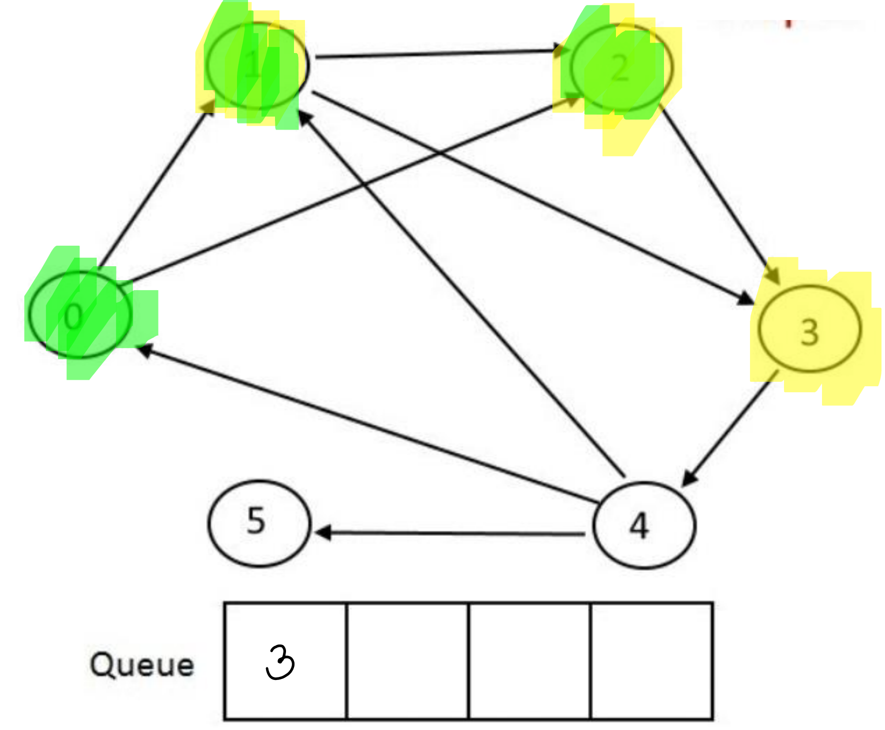
В ширину: BFS

- 1. Помечаем все вершины как не пройденные
- 2. Добавляем в очередь стартовую вершину
- 3. В цикле: (пока очередь не пустая)
  - a. Берем вершину из очереди и помечаем ее как пройдённую
  - b. Добавляем в очередь все смежные с ней вершины, которые не пройденные
  - c. Удаляем из очереди пройденную вершину



В ширину: BFS

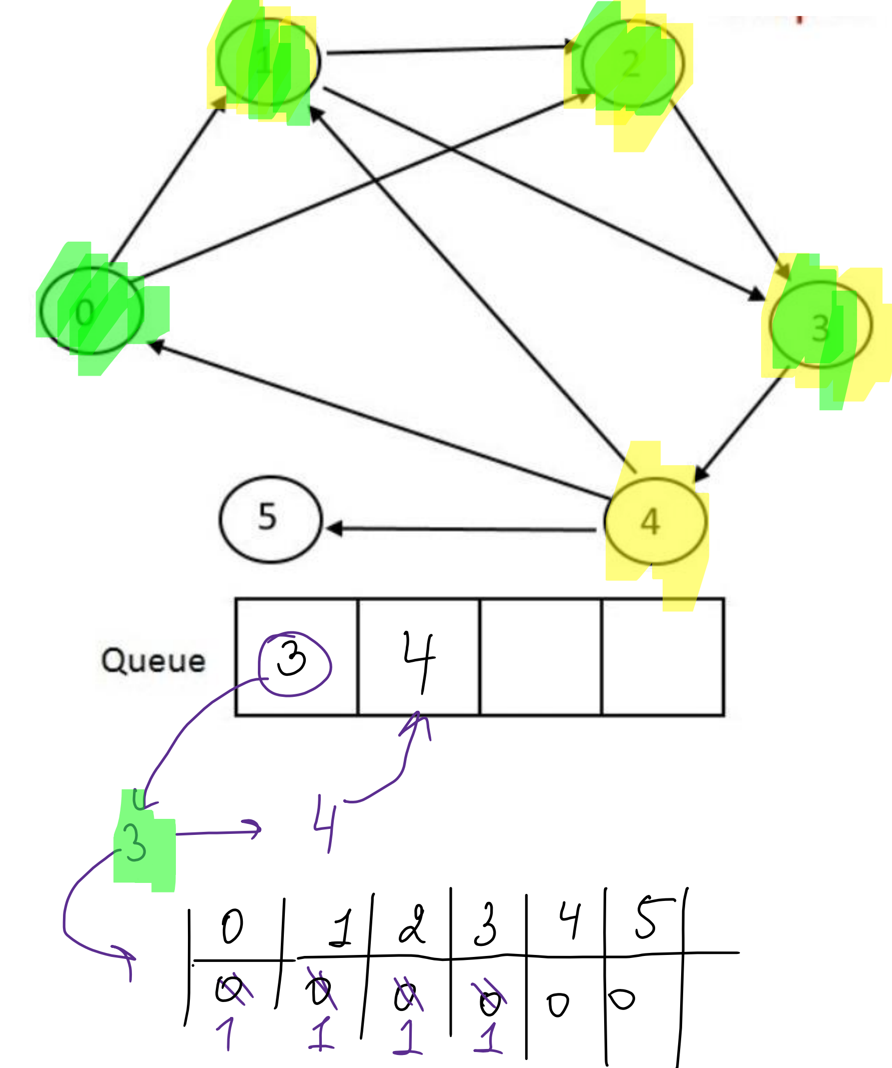
- 1. Помечаем все вершины как не пройденные
- 2. Добавляем в очередь стартовую вершину
- 3. В цикле: (пока очередь не пустая)
  - a. Берем вершину из очереди и помечаем ее как пройдённую
  - b. Добавляем в очередь все смежные с ней вершины, которые не пройденные
  - c. Удаляем из очереди пройденную вершину



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

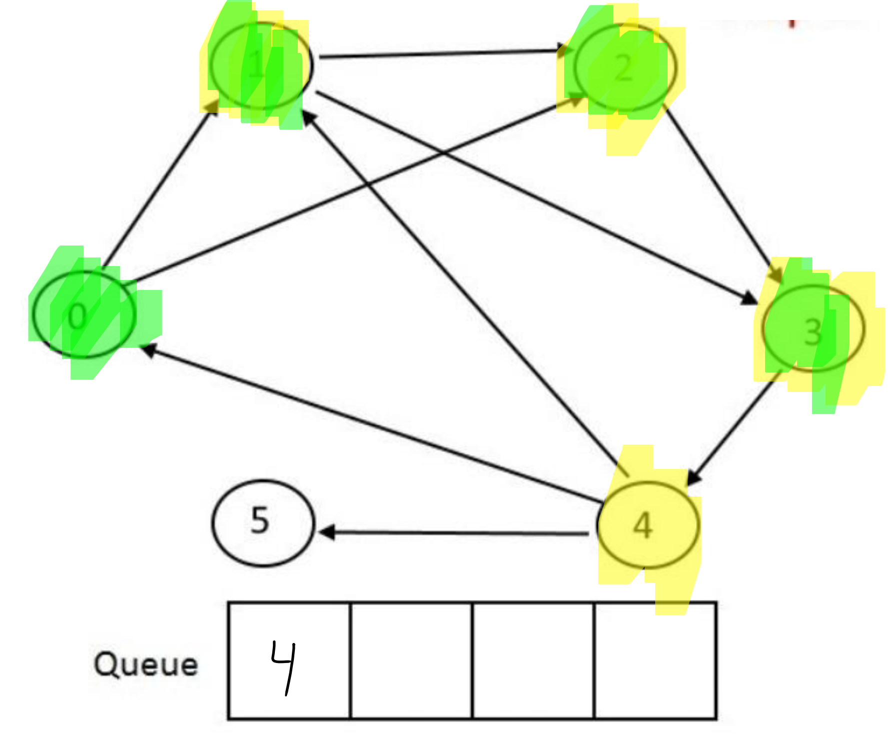
В ширину: BFS

- 1. Помечаем все вершины как не пройденные
- 2. Добавляем в очередь стартовую вершину
- 3. В цикле: (пока очередь не пустая)
    - a. Берем вершину из очереди и помечаем ее как пройдённую
    - b. Добавляем в очередь все смежные с ней вершины, которые не пройденные
    - c. Удаляем из очереди пройденную вершину



В ширину: BFS

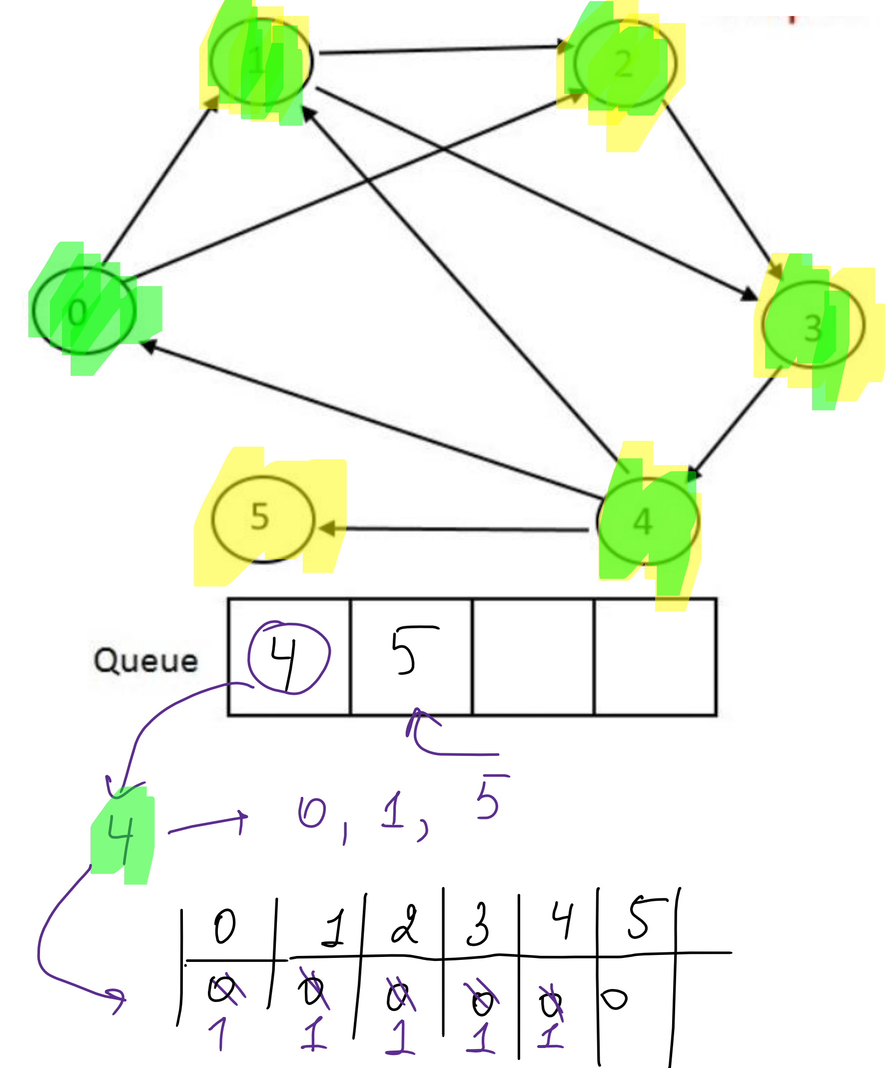
- 1. Помечаем все вершины как не пройденные
- 2. Добавляем в очередь стартовую вершину
- 3. В цикле: (пока очередь не пустая)
  - a. Берем вершину из очереди и помечаем ее как пройдённую
  - b. Добавляем в очередь все смежные с ней вершины, которые не пройденные
  - c. Удаляем из очереди пройденную вершину



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

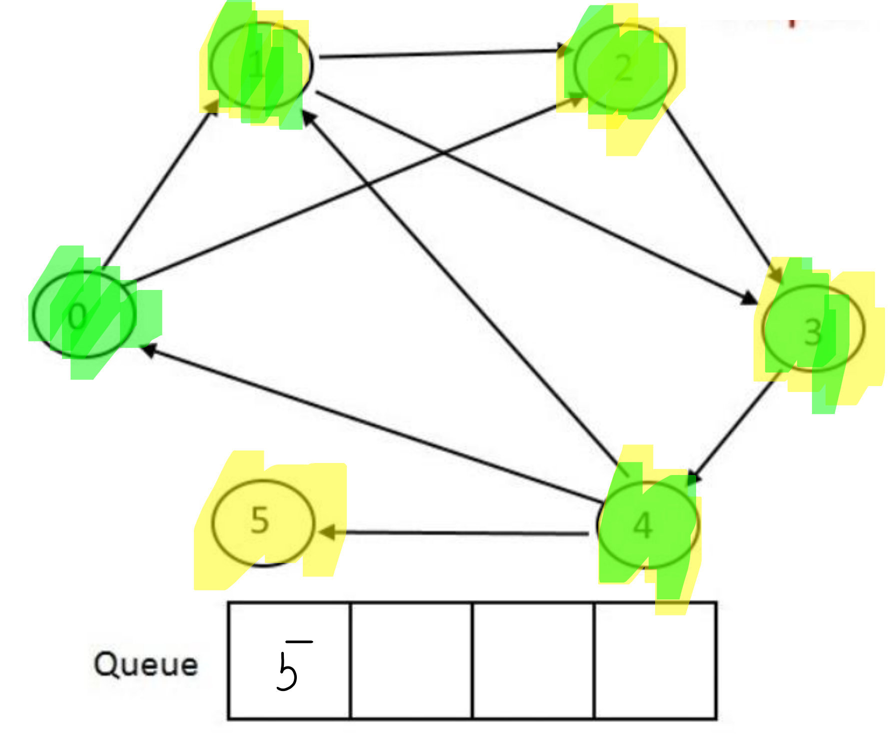
В ширину: BFS

- 1. Помечаем все вершины как не пройденные
- 2. Добавляем в очередь стартовую вершину
- 3. В цикле: (пока очередь не пустая)
  - a. Берем вершину из очереди и помечаем ее как пройдённую
  - b. Добавляем в очередь все смежные с ней вершины, которые не пройденные
  - c. Удаляем из очереди пройденную вершину



В ширину: BFS

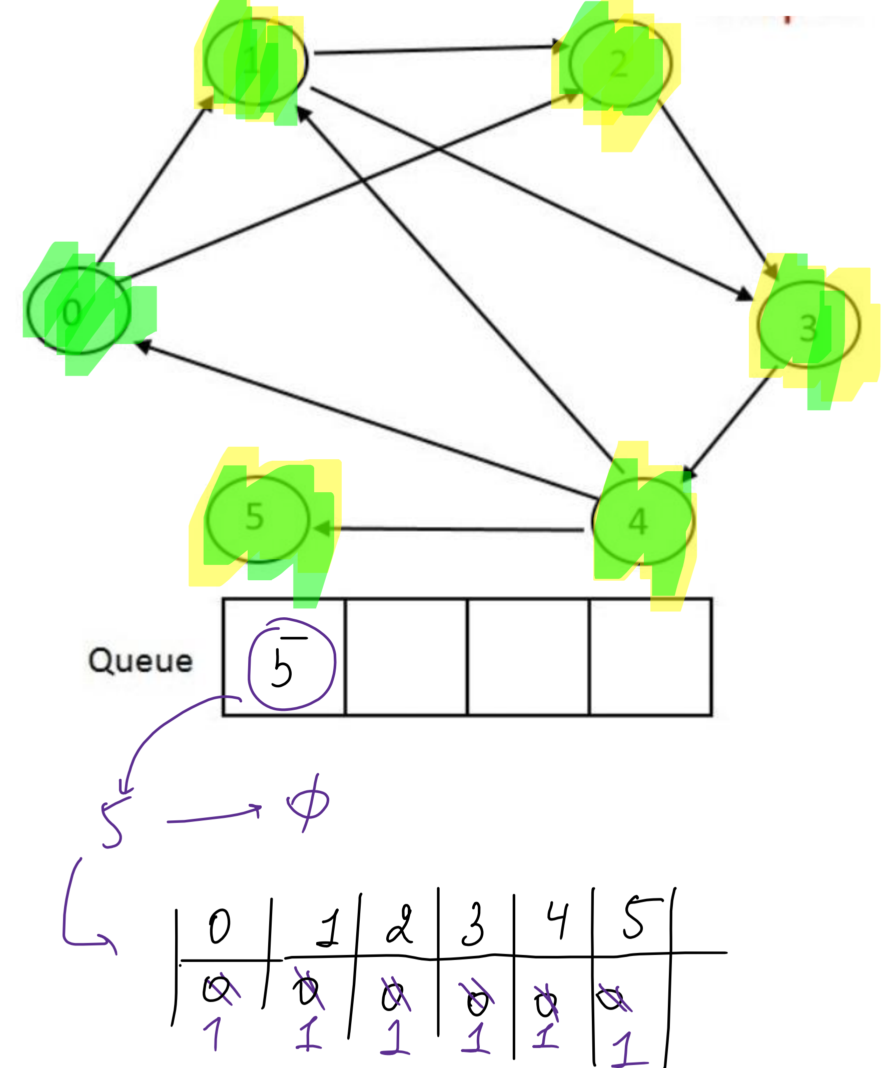
- 1. Помечаем все вершины как не пройденные
- 2. Добавляем в очередь стартовую вершину
- 3. В цикле: (пока очередь не пустая)
    - a. Берем вершину из очереди и помечаем ее как пройдённую
    - b. Добавляем в очередь все смежные с ней вершины, которые не пройденные
    - c. Удаляем из очереди пройденную вершину



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

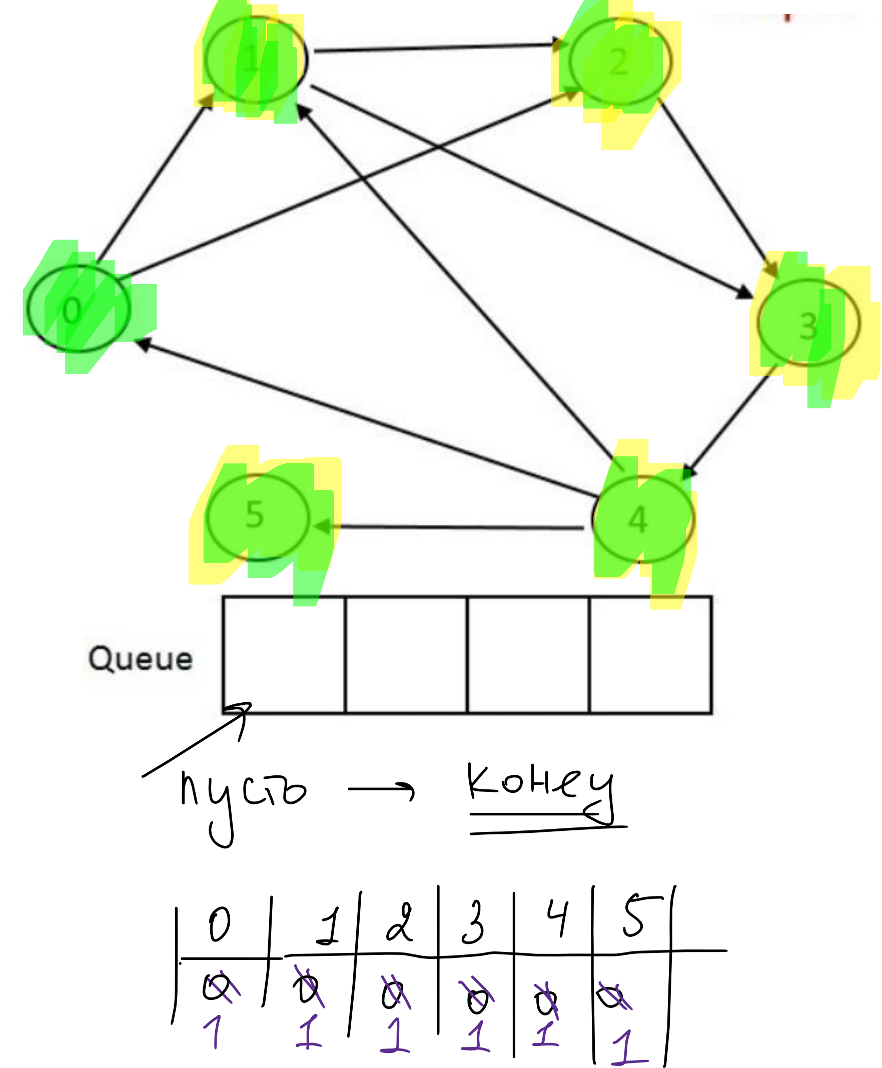
В ширину: BFS

- 1. Помечаем все вершины как не пройденные
- 2. Добавляем в очередь стартовую вершину
- 3. В цикле: (пока очередь не пустая)
  - a. Берем вершину из очереди и помечаем ее как пройдённую
  - b. Добавляем в очередь все смежные с ней вершины, которые не пройденные
  - c. Удаляем из очереди пройденную вершину



В ширину: BFS

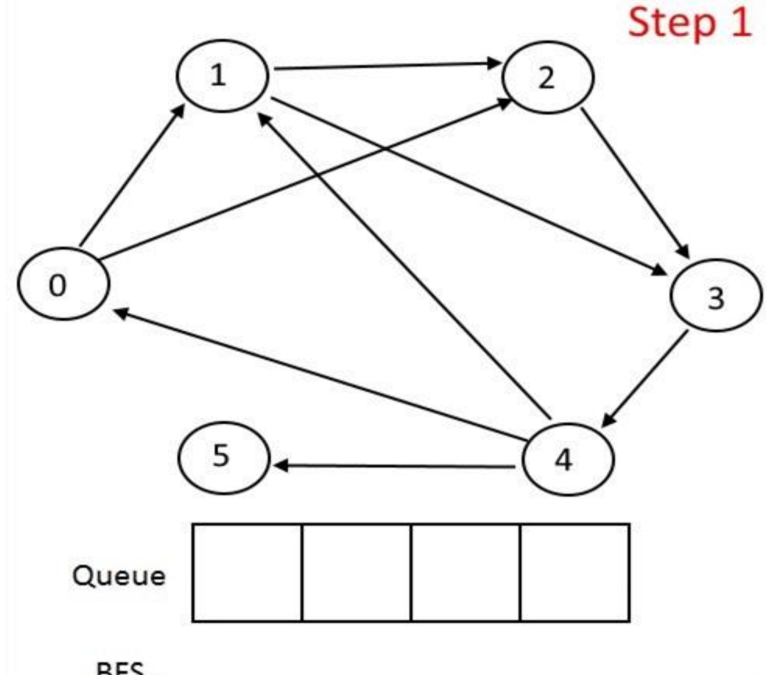
- 1. Помечаем все вершины как не пройденные
- 2. Добавляем в очередь стартовую вершину
- 3. В цикле: (пока очередь не пустая)
    - a. Берем вершину из очереди и помечаем ее как пройдённую
    - b. Добавляем в очередь все смежные с ней вершины, которые не пройденные
    - c. Удаляем из очереди пройденную вершину



Асимптотика: O(|V|+|E|)

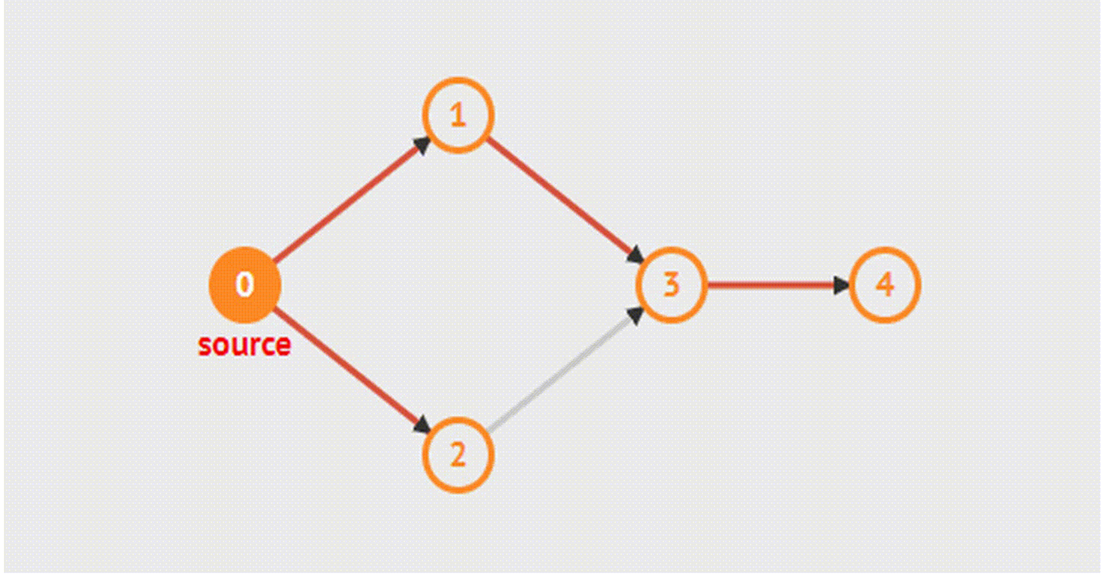
Оценим время работы для входного графа G=(V,E), где множество ребер E представлено списком смежности.

- В очередь добавляются только непосещенные вершины, поэтому каждая вершина посещается не более одного раза.
- Операции внесения в очередь и удаления из нее требуют O(1) времени, так что общее время работы с очередью составляет O(|V|) операций.
- Для каждой вершины v рассматривается не более deg(v) ребер, инцидентных ей.
- Так как ∑ deg(v)=2|E|, то время, используемое на работу с ребрами, составляет O(|E|).

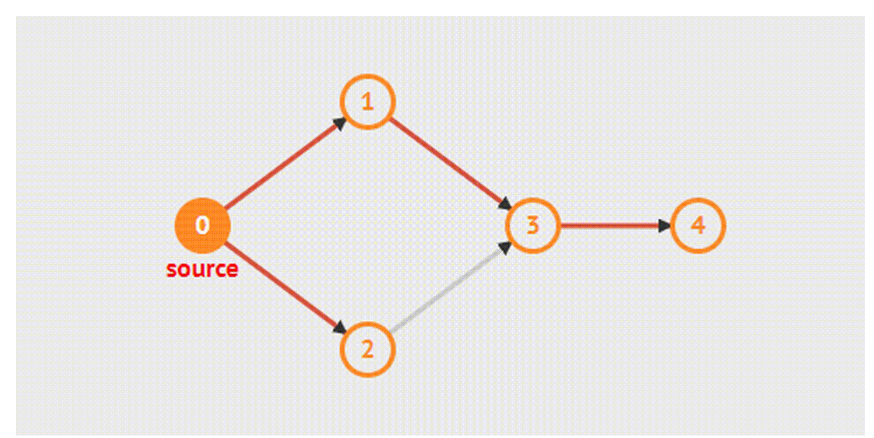


Поиск в ширину

```
1.int BFS(G: (V, E), source: int, destination: int):
2.     d = int[|V|]
3.     fill(d, ∞)
4.     d[source] = 0
5.     Q = ∅
6.     Q.push(source)
7.     while Q ≠∅
8.         u = Q.pop()
9.         for v: (u, v) in E
10.             if d[v] == ∞
11.                 d[v] = d[u] + 1
12.                 Q.push(v)
13.     return d[destination]
```



Дерево поиска в ширину



Поиск в ширину также может построить дерево поиска в ширину.

- Изначально оно состоит из одного корня s.
- Когда мы добавляем непосещенную вершину в очередь, то добавляем ее и ребро, по которому мы до нее дошли, в дерево.
- Поскольку каждая вершина может быть посещена не более одного раза, она имеет не более одного родителя.
- После окончания работы алгоритма для каждой достижимой из s вершины t путь в дереве поиска в ширину соответствует кратчайшему пути от s до t в G.

Обход в ширину

- Как найти расстояние до всех вершин от стартовой ?
  - *Использовать дополнительный массив для подсчета расстояния от стартовой до всех остальных*
  - *+ Массив предков для восстановления путей*

Обход в ширину

- <u>Как найти расстояние до всех вершин от стартовой ?</u>
  - *Использовать дополнительный массив для подсчета расстояния от стартовой до всех остальных*
  - *+ Массив предков для восстановления путей*

$d[v] = \infty$ – на старте, $d[s] = 0$ – от стартовой до самой себя

parent[ ] = null → какая вершина мой родитель

parent!

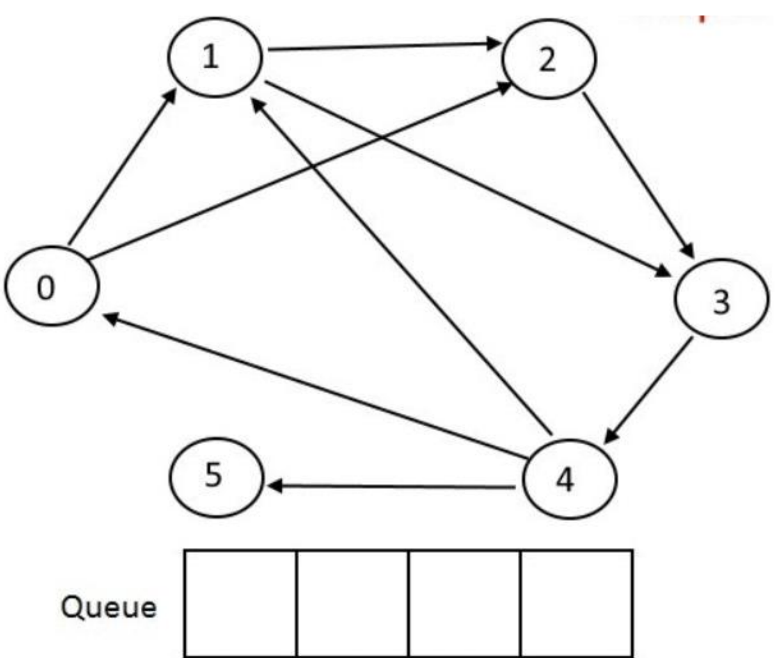
S (0) → 1 → 2 → 3
2 → 4 → 5

0	1	2	3	4	5
∞	0	1	2	2	4

путь от 0 до 5 : 5 → 4 → 2 → 1 → 0 (восстановлен)

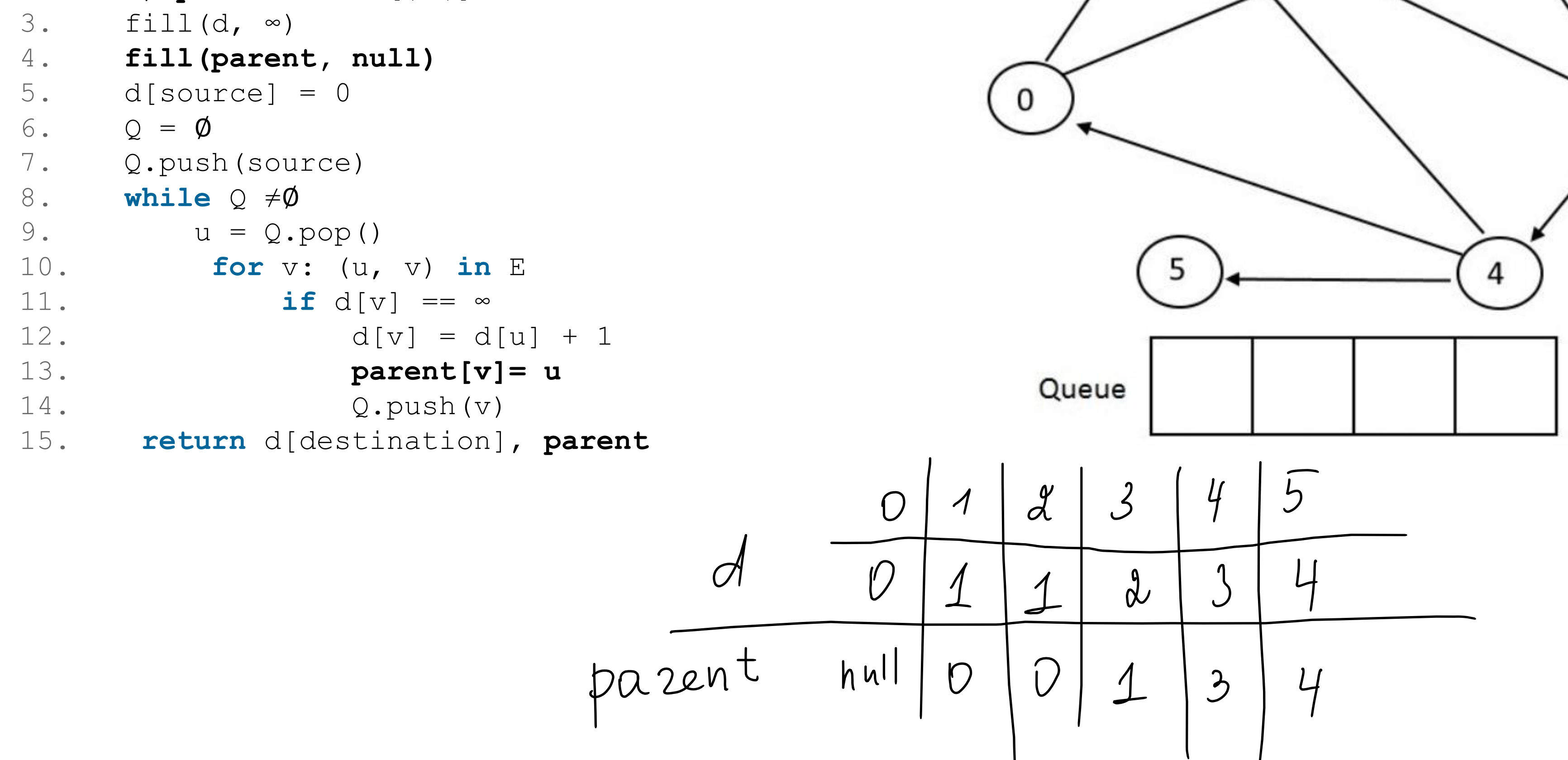
Поиск в ширину

```
1.int BFS(G: (V, E), source: int, destination: int):
2.      d, parent = int[|V|]
3.      fill(d, ∞)
4.      fill(parent, null)
5.      d[source] = 0
6.      Q = Ø
7.      Q.push(source)
8.      while Q ≠Ø
9.          u = Q.pop()
10.          for v: (u, v) in E
11.              if d[v] == ∞
12.                  d[v] = d[u] + 1
13.                  parent[v]= u
14.                  Q.push(v)
15.      return d[destination], parent
```



Поиск в ширину

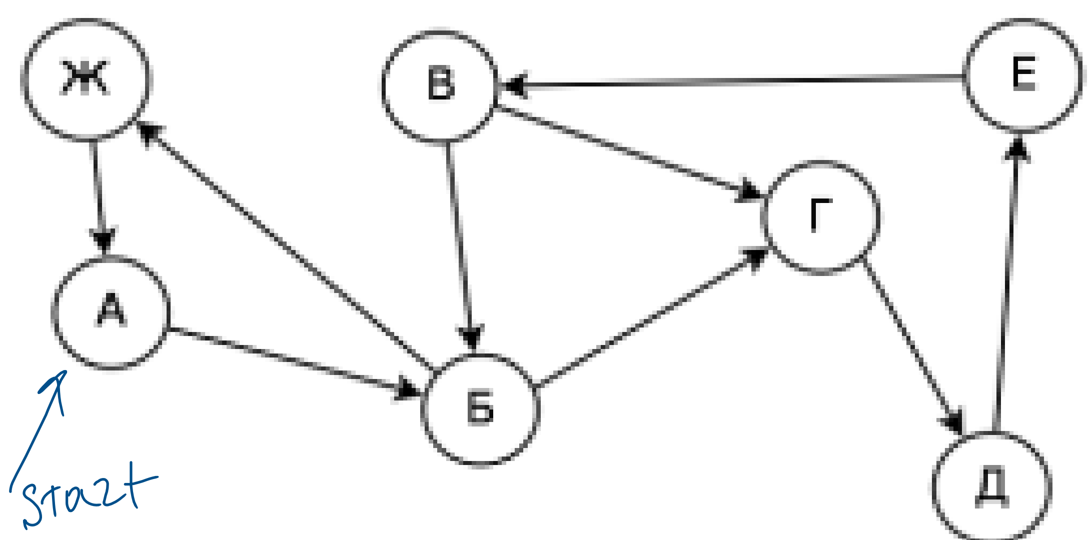
```
1.int BFS(G: (V, E), source: int, destination: int):
2.      d, parent = int[|V|]
3.      fill(d, ∞)
4.      fill(parent, null)
5.      d[source] = 0
6.      Q = ∅
7.      Q.push(source)
8.      while Q ≠∅
9.          u = Q.pop()
10.          for v: (u, v) in E
11.              if d[v] == ∞
12.                  d[v] = d[u] + 1
13.                  parent[v]= u
14.                  Q.push(v)
15.      return d[destination], parent
```



	0	1	2	3	4	5
d	0	1	1	2	3	4
parent	null	0	0	1	3	4

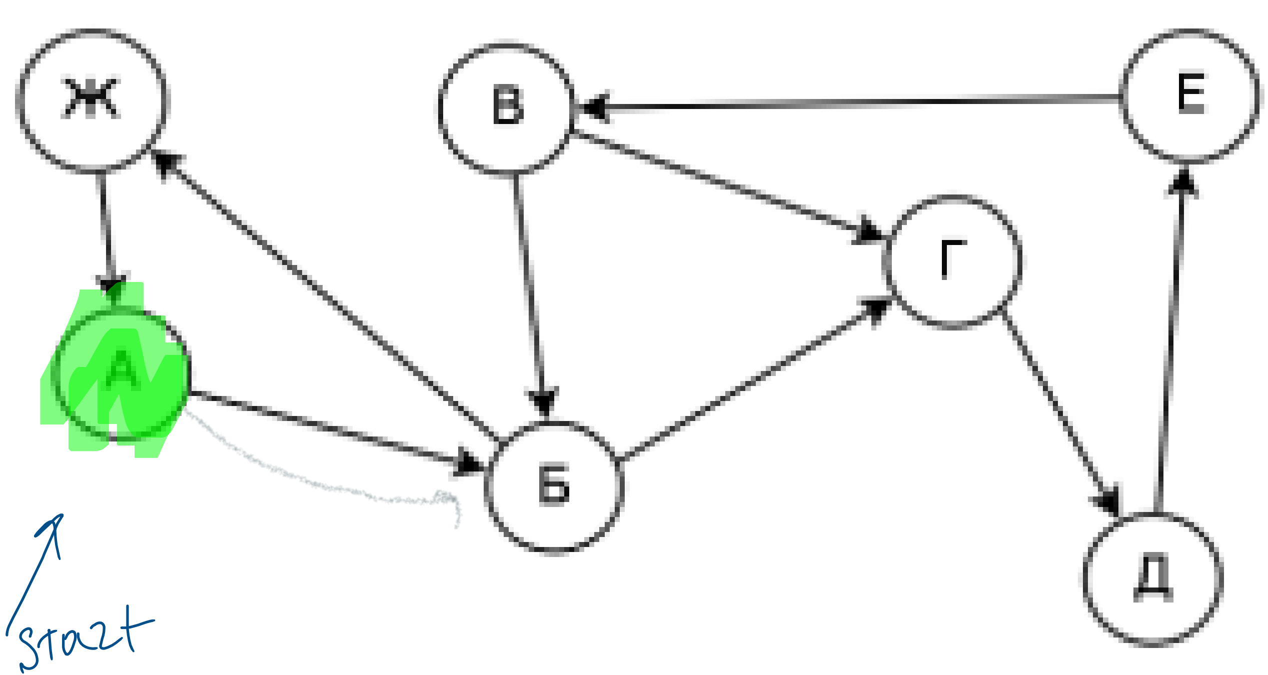
Обход в ГЛУБИНУ

- Идем от вершины к вершине по смежным и помечаем как пройденные пока не пройдем все возможные!



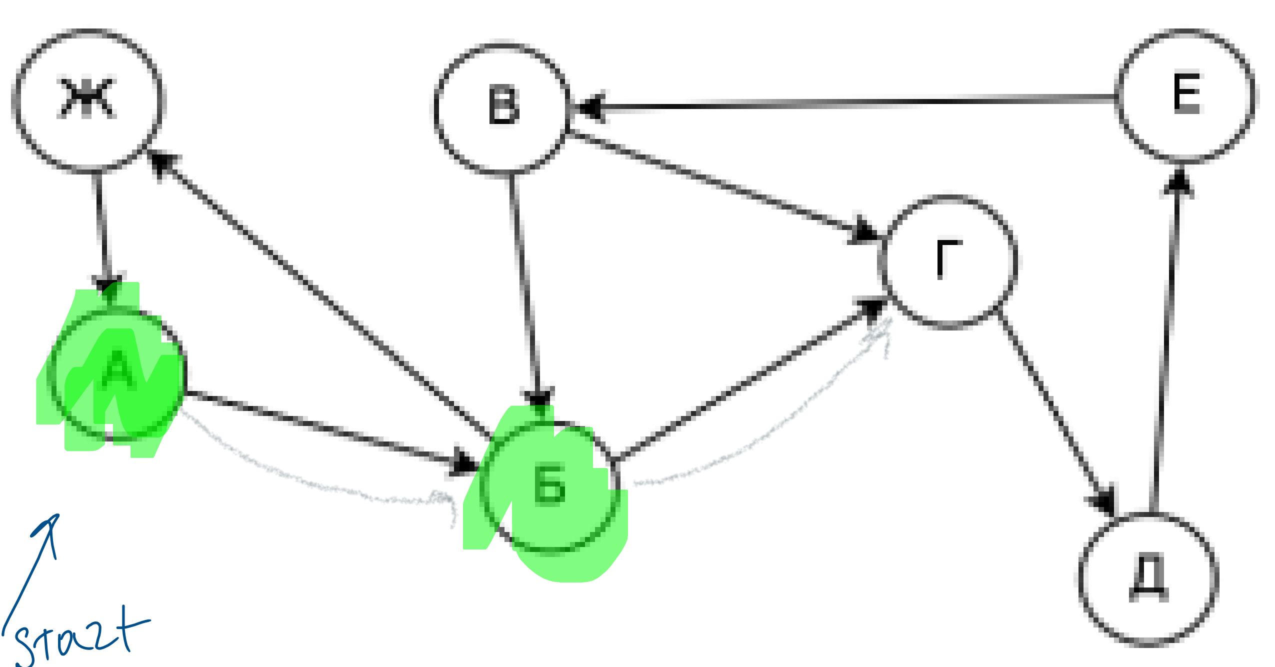
Обход в ГЛУБИНУ

- Идем от вершины к вершине по смежным и помечаем как пройденные пока не пройдем все возможные!



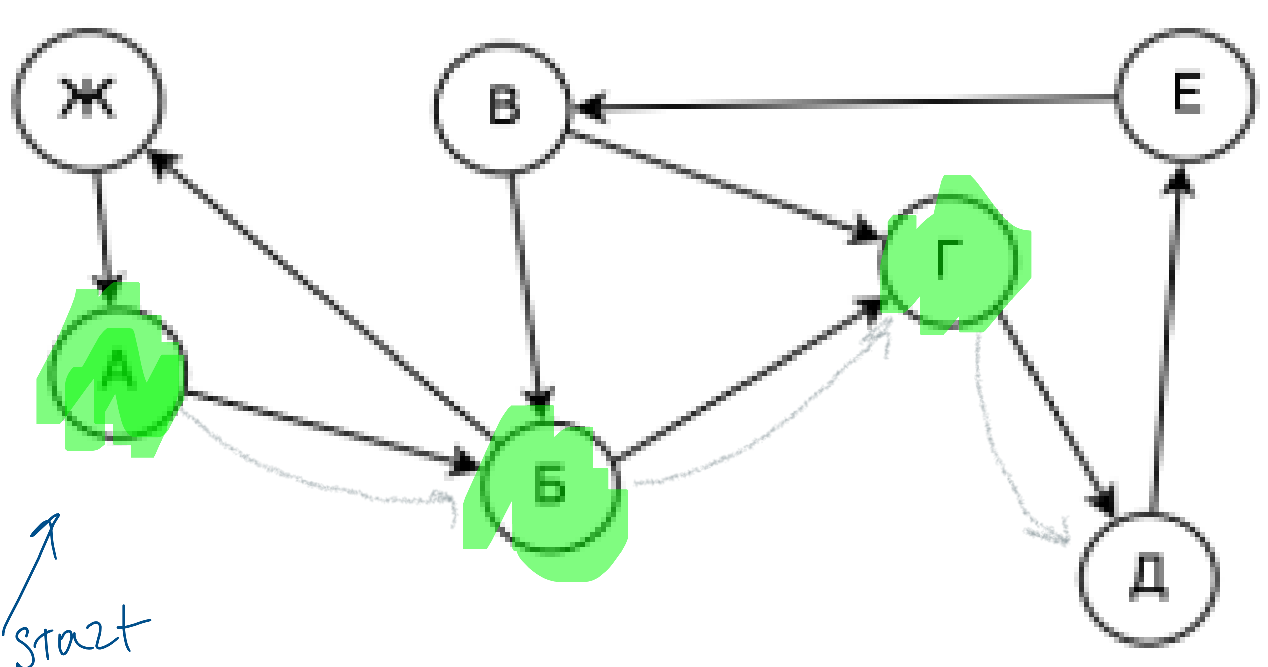
Обход в ГЛУБИНУ

- Идем от вершины к вершине по смежным и помечаем как пройденные пока не пройдем все возможные!



Обход в ГЛУБИНУ

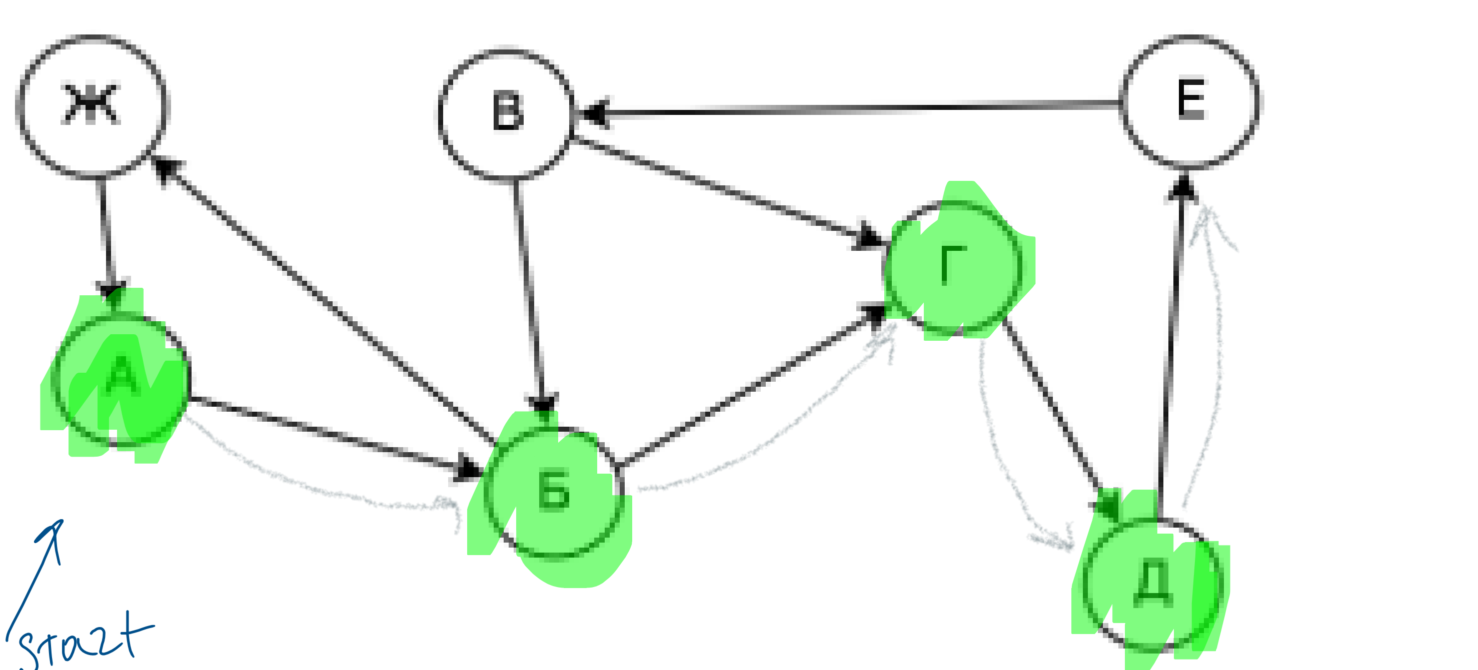
- Идем от вершины к вершине по смежным и помечаем как пройденные пока не пройдем все возможные!



А
Б
Г

Обход в ГЛУБИНУ

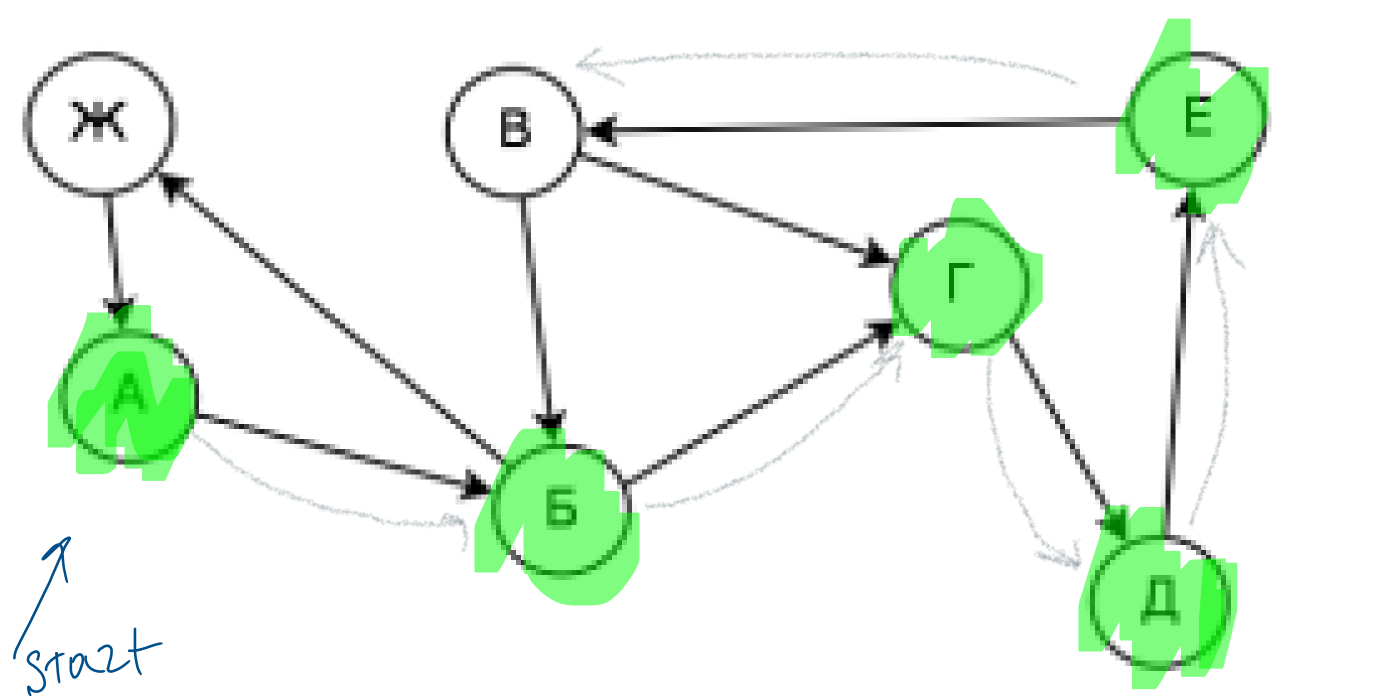
- Идем от вершины к вершине по смежным и помечаем как пройденные пока не пройдем все возможные!



А
Б
Г
Д

Обход в ГЛУБИНУ

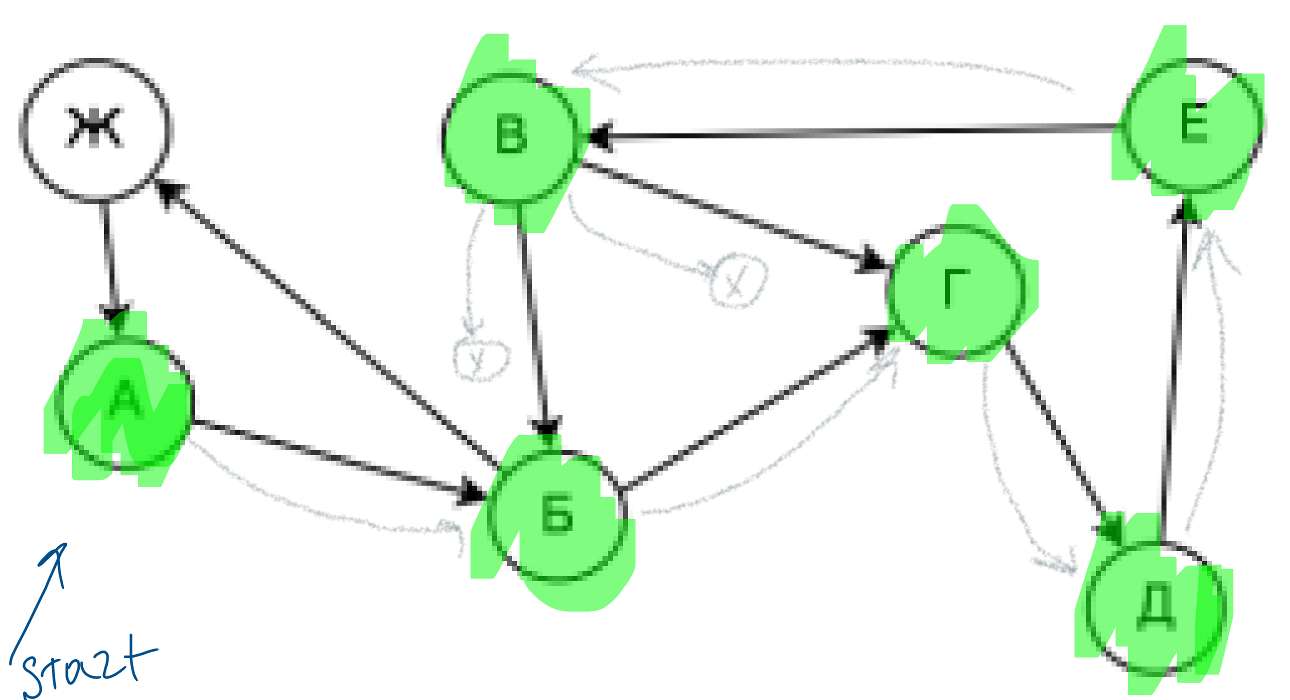
- Идем от вершины к вершине по смежным и помечаем как пройденные пока не пройдем все возможные!



А
Б
Г
Д
Е

Обход в ГЛУБИНУ

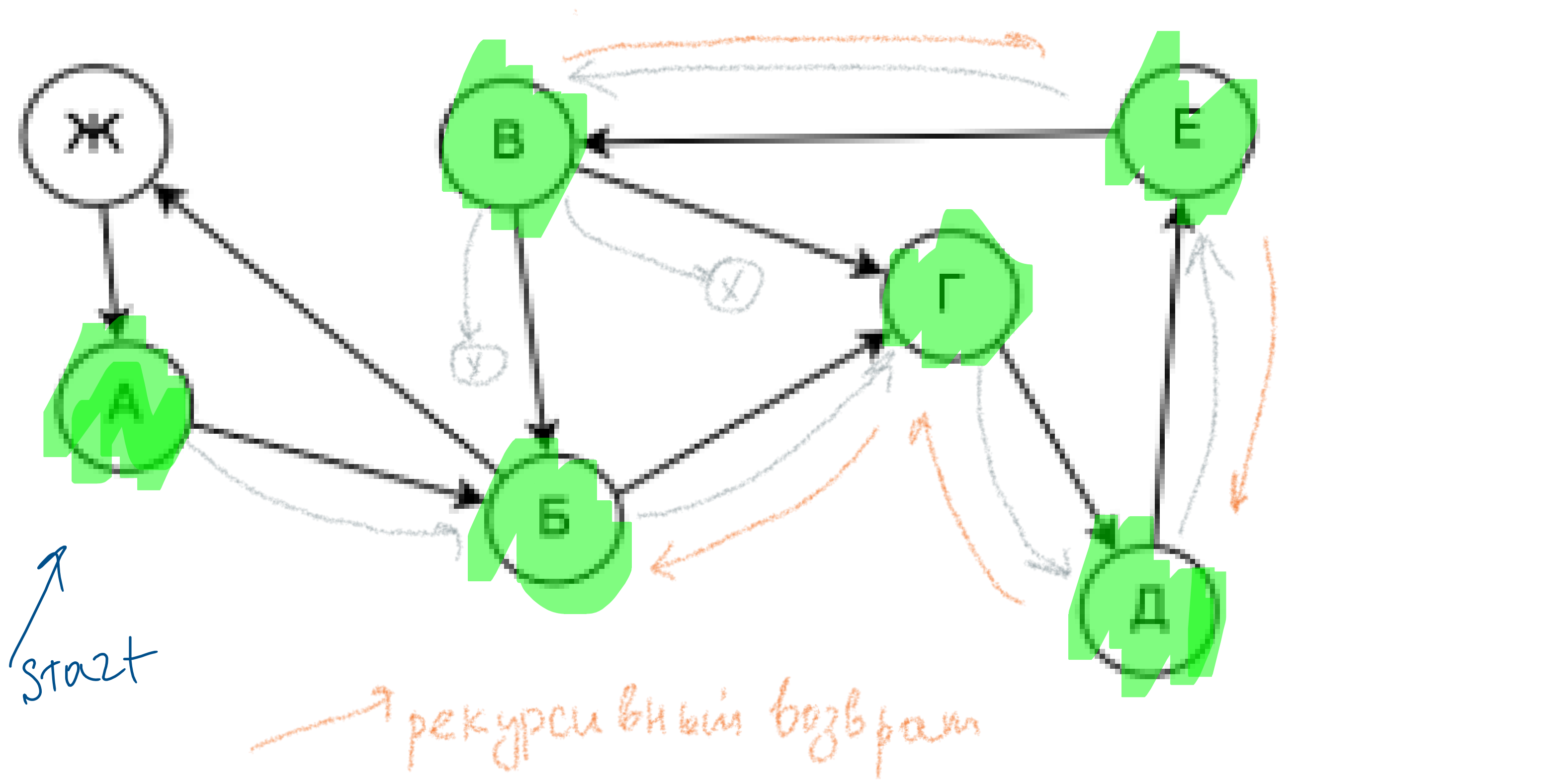
- Идем от вершины к вершине по смежным и помечаем как пройденные пока не пройдем все возможные!



А
Б
Г
Д
Е
В

Обход в ГЛУБИНУ

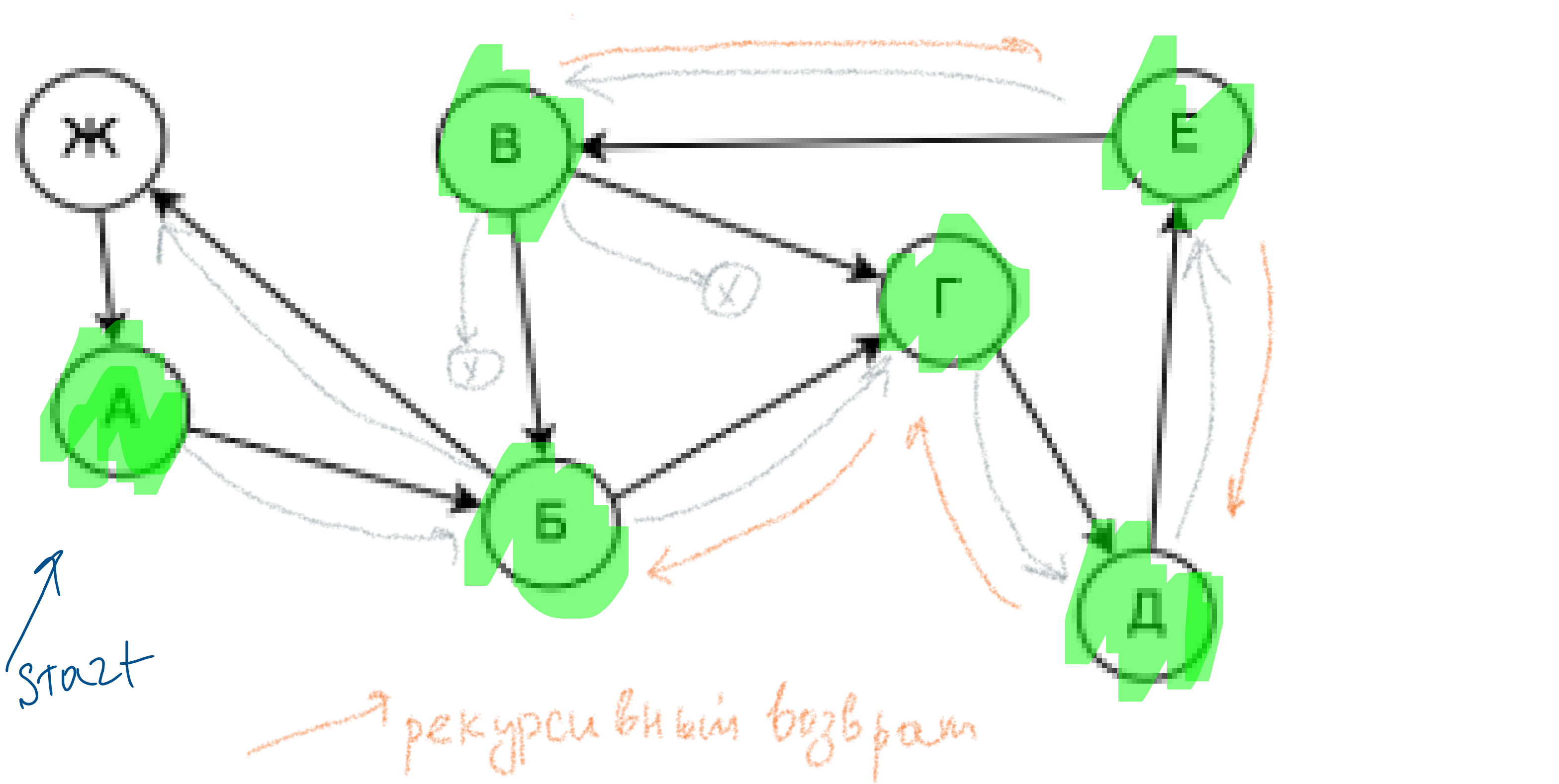
- Идем от вершины к вершине по смежным и помечаем как пройденные пока не пройдем все возможные!



А
Б
Г
Д
Е
В

Обход в ГЛУБИНУ

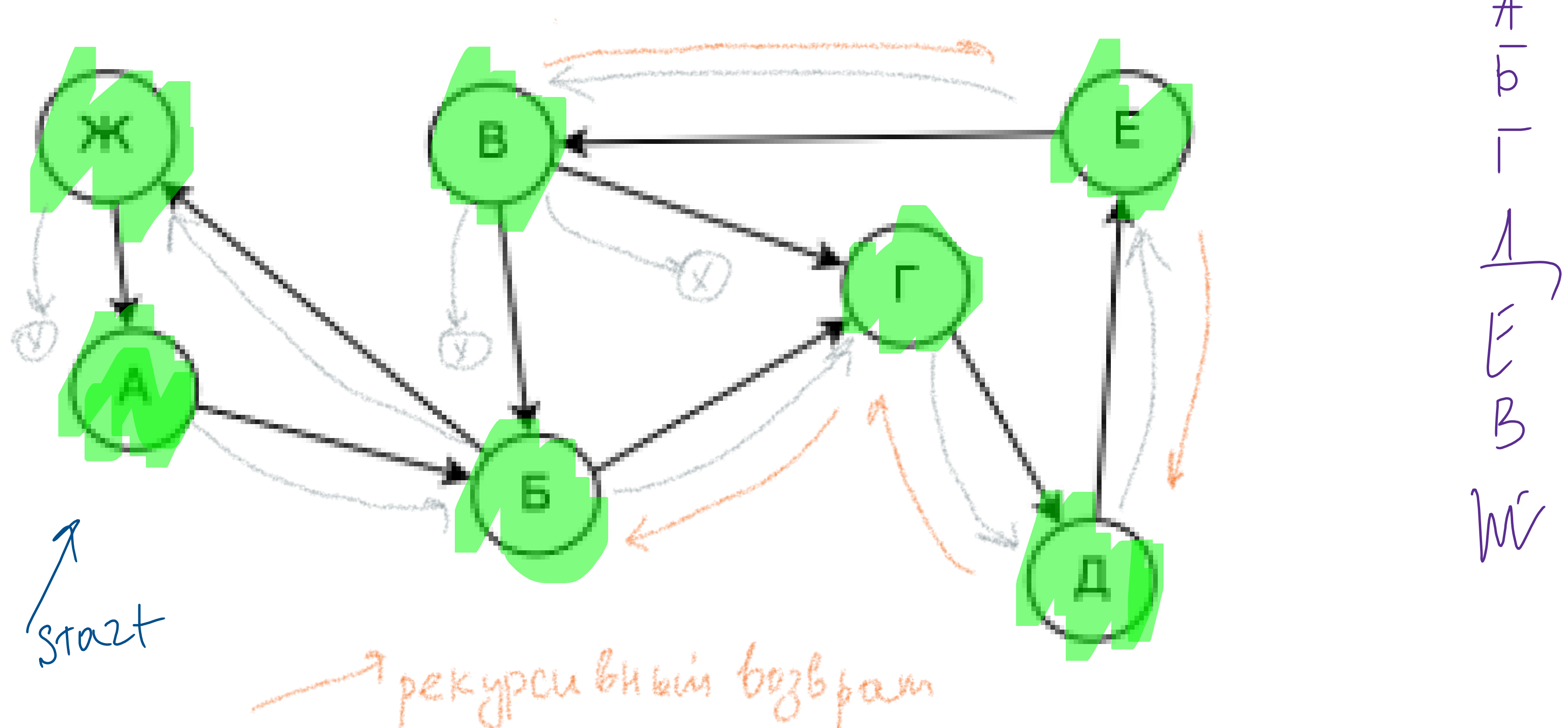
- Идем от вершины к вершине по смежным и помечаем как пройденные пока не пройдем все возможные!



А
Б
Г
Д
Е
В

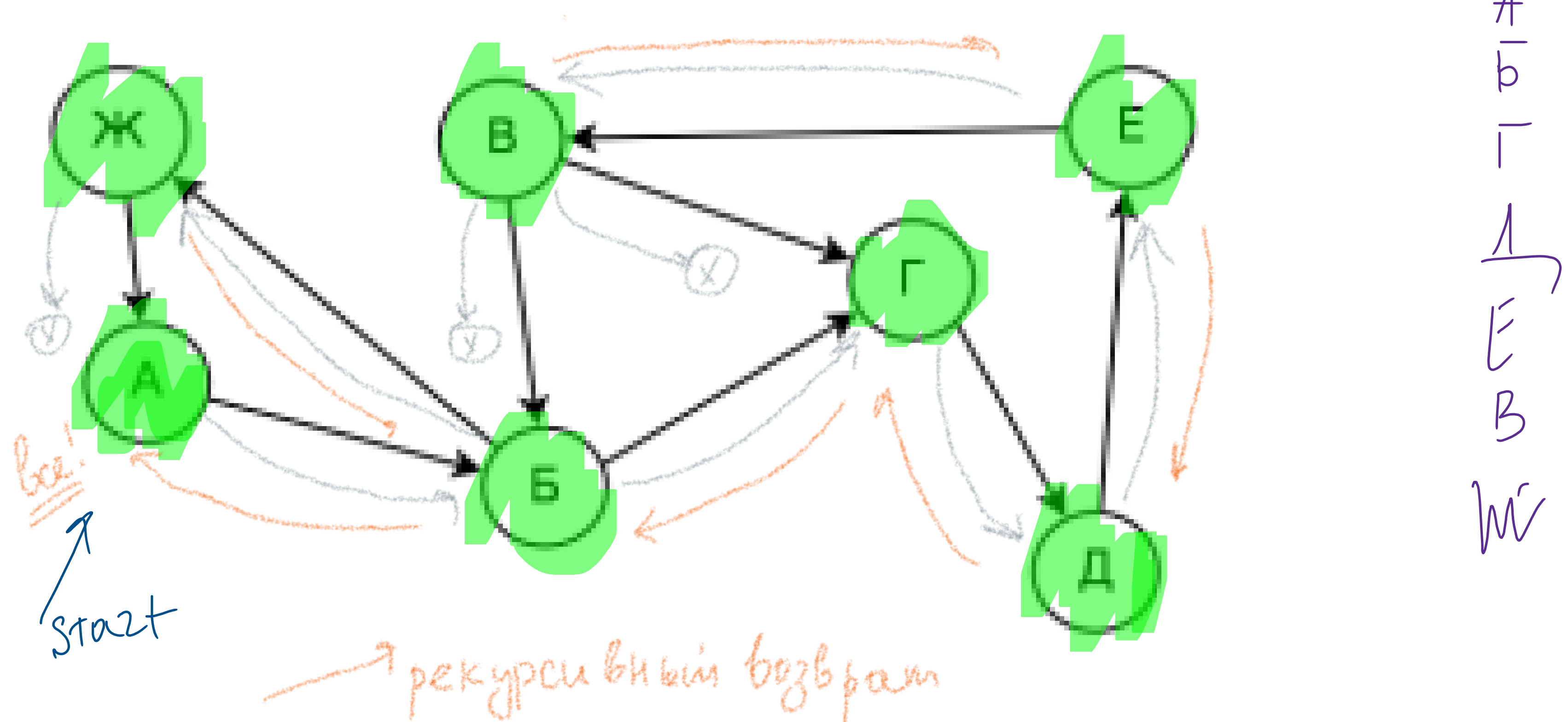
Обход в ГЛУБИНУ

- Идем от вершины к вершине по смежным и помечаем как пройденные пока не пройдем все возможные!



Обход в ГЛУБИНУ

- Идем от вершины к вершине по смежным и помечаем как пройденные пока не пройдем все возможные!



Обход в глубину

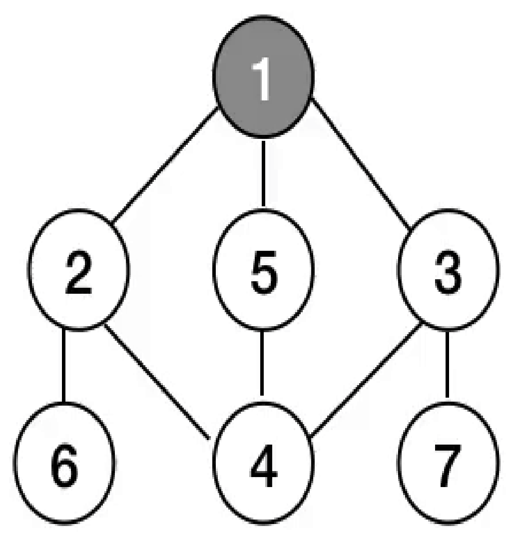
- Какие идеи эффективной реализации?
- Как понять, что алгоритм закончен?
- Как лучше хранить граф?

Обход в глубину

- <u>Какие идеи эффективной реализации?</u>
  - *Использовать рекурсию для прохода по графу*
  - *Использовать метки – пройдена/не пройдена*
  - *Рассматриваем в один момент времени – одну вершину*
- <u>Как понять, что алгоритм закончен?</u>
  - *Все вершины помечены как пройденные, даже если несколько компонент связности*
- <u>Как лучше хранить граф?</u>
  - *Список смежности/Матрица смежности*

Обход в глубину: DFS (G(V, E))

```
1.DFS (G) {
2.     For  u из G.V do
3.         u.color = white
4.     For  u из G.V do
5.     If  u.color == white  then
6.             Visit (G, u)
7.}
8.
9.Visit (G) {
10.     u.color = gray
11.     For  v  из G.V[u]  do
12.         If  v.color == white  then
13.             Visit (G, v)
14.     u.color = black
15.}
```



1

white → не пройдена
gray → в процессе dfs
black → пройдена

Обход в глубину: DFS (G(V, E))

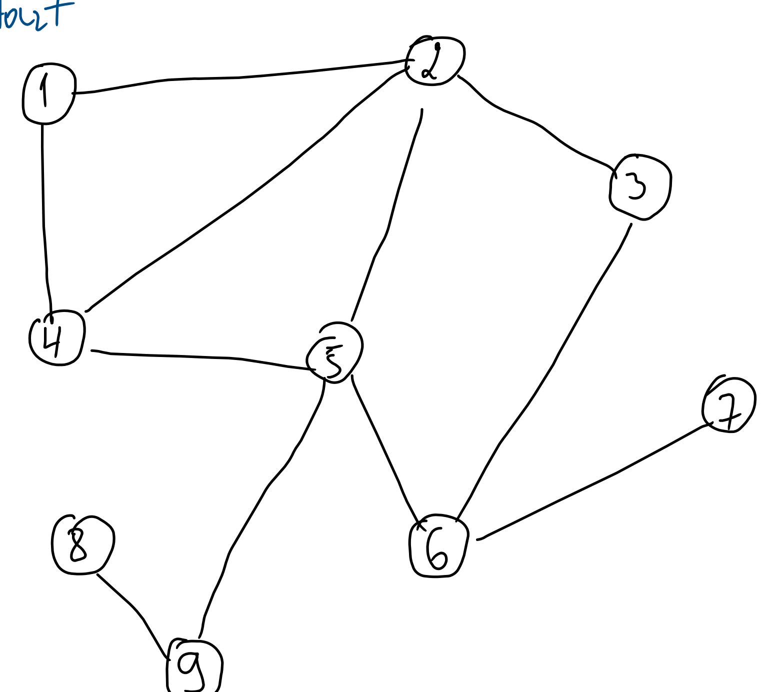
```
1.DFS (G) {
2.     For  u из G.V do
3.         u.color = white
4.     For  u из G.V do
5.     If  u.color == white  then
6.             Visit (G, u)
7.}
8.
9.Visit (G) {
10.     u.color = gray
11.     For  v  из G.V[u]  do
12.         If  v.color == white  then
13.             Visit (G, v)
14.     u.color = black
15.}
```

сперва все не пройдены

все пройдём

white → не пройдена
gray → в процессе dfs
black → пройдена

Обход в ГЛУБИНУ: три цвета



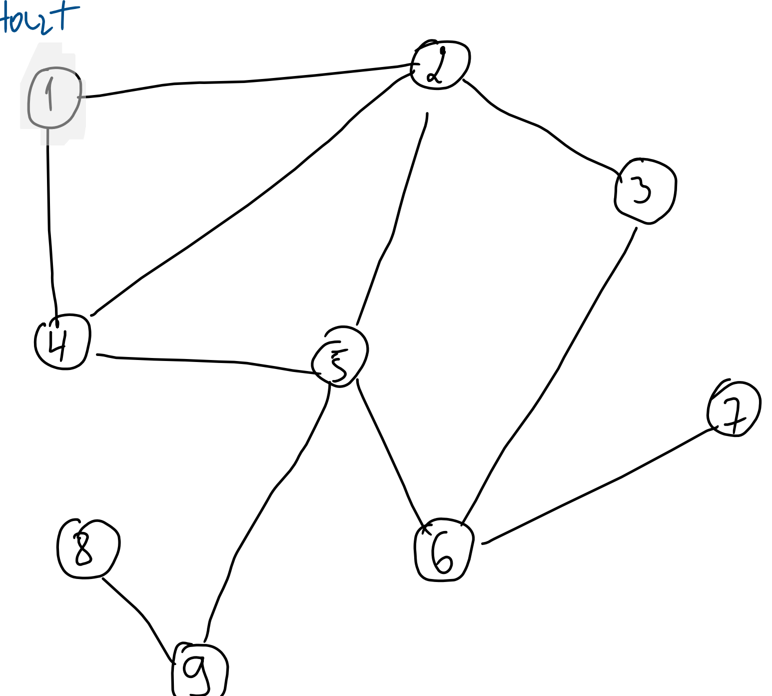
не пройдена

в процессе прохода

пройдена

```
1.DFS (G) {
2.    For  u из G.V do
3.        u.color = white
4.    For  u из G.V do
5.    If  u.color == white  then
6.            Visit (G, u)
7.}
8.
9.Visit (G) {
10.    u.color = gray
11.    For  v  из G.V[u]  do
12.        If  v.color == white  then
13.            Visit (G, v)
14.    u.color = black
15.}
```

Обход в ГЛУБИНУ: три цвета



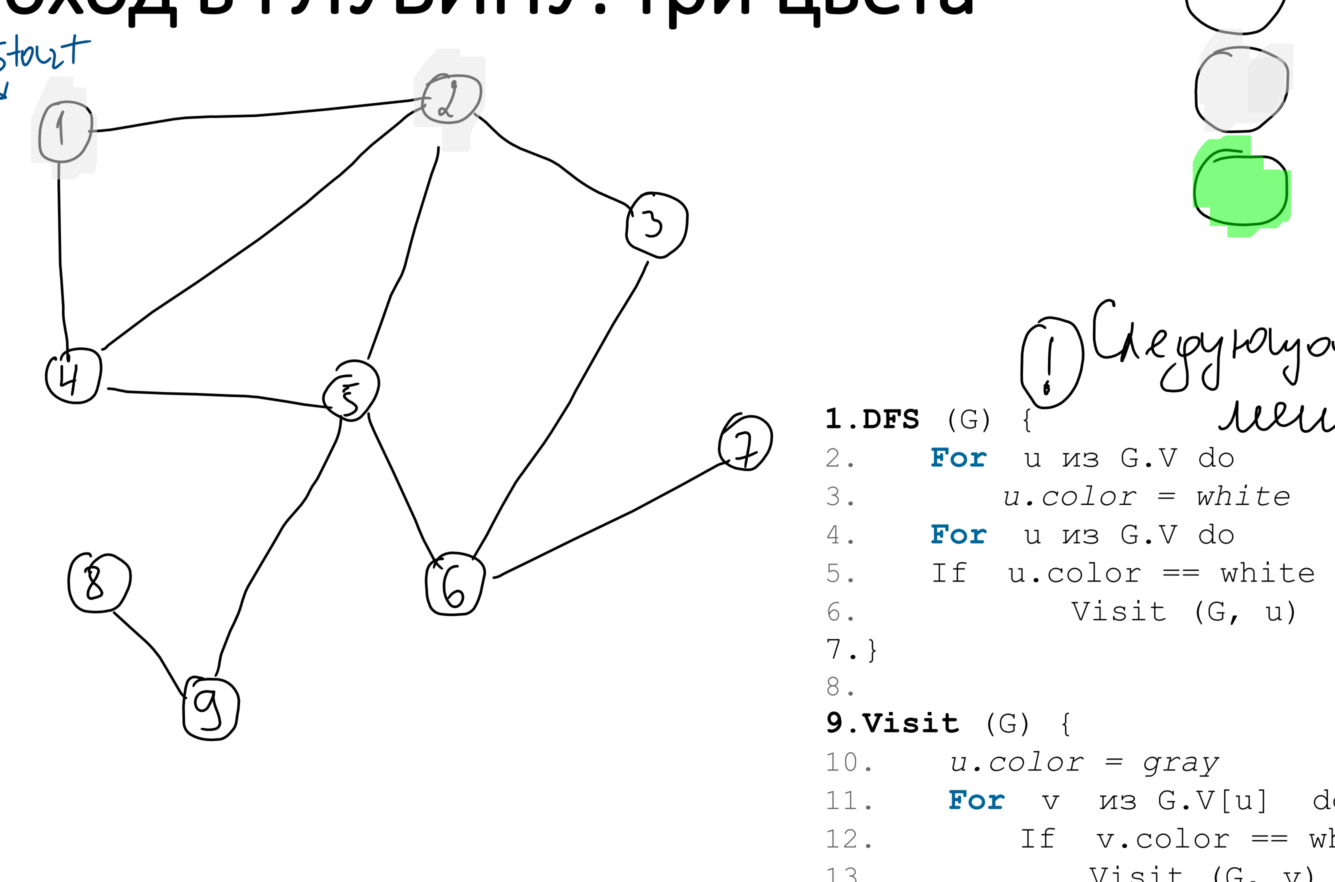
не пройдена

в процессе прохода

пройдена

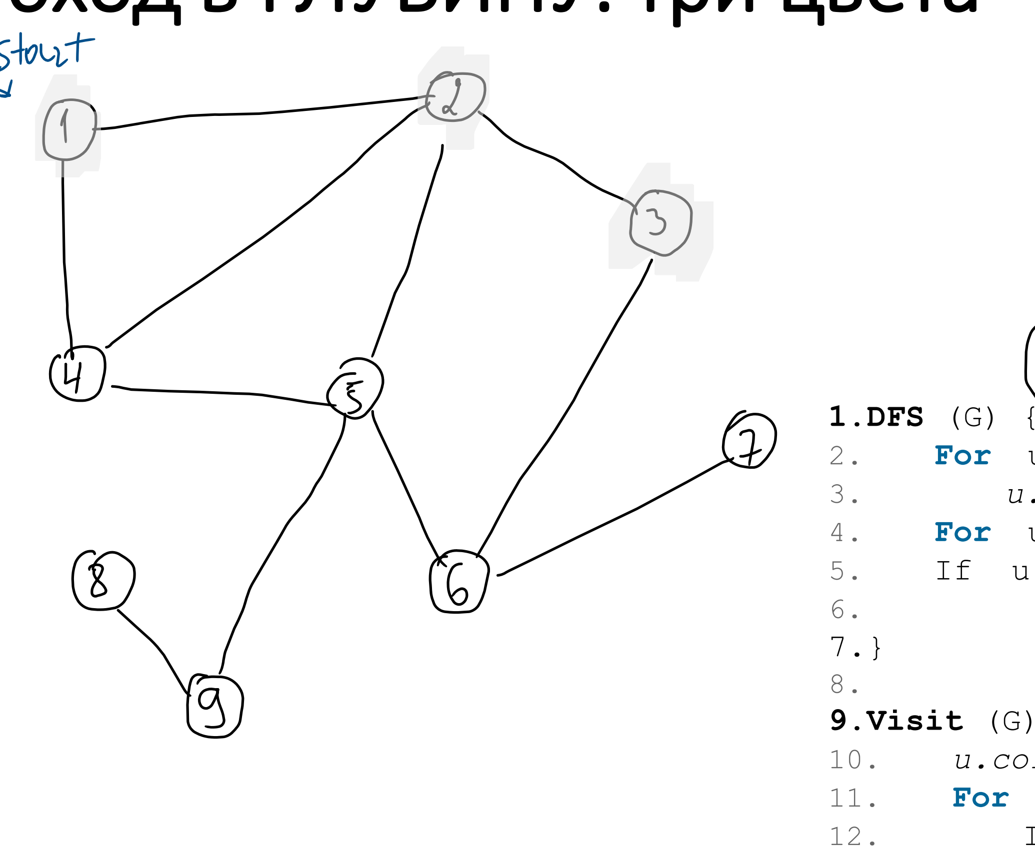
```
1.DFS (G) {
2.     For  u из G.V do
3.          u.color = white
4.     For  u из G.V do
5.     If  u.color == white  then
6.              Visit (G, u)
7.}
8.
9.Visit (G) {
10.    u.color = gray
11.    For  v  из G.V[u]  do
12.         If  v.color == white  then
13.             Visit (G, v)
14.    u.color = black
15.}
```

Обход в ГЛУБИНУ: три цвета




```
1.DFS (G) {
2.    For  u из G.V do
3.        u.color = white
4.    For  u из G.V do
5.    If  u.color == white  then
6.            Visit (G, u)
7.}
8.
9.Visit (G) {
10.    u.color = gray
11.    For  v  из G.V[u]  do
12.        If  v.color == white  then
13.            Visit (G, v)
14.    u.color = black
15.}
```

Обход в ГЛУБИНУ: три цвета



не пройдена

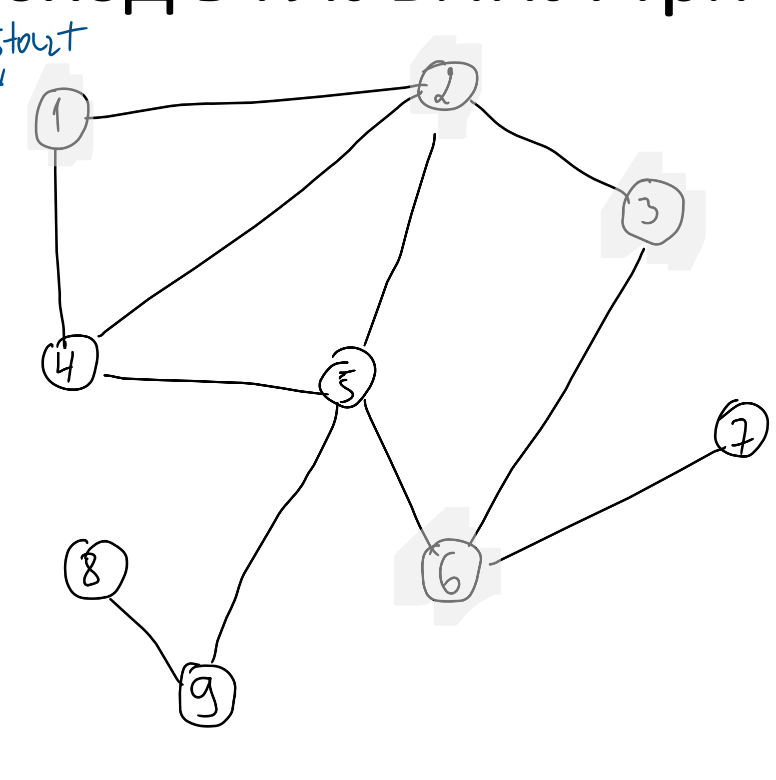
в процессе прохода

пройдена

! Следующая всегда с меньшим номером

```
1.DFS (G) {
2.    For  u из G.V do
3.        u.color = white
4.    For  u из G.V do
5.    If  u.color == white  then
6.            Visit (G, u)
7.}
8.
9.Visit (G) {
10.    u.color = gray
11.    For  v  из G.V[u]  do
12.        If  v.color == white  then
13.            Visit (G, v)
14.    u.color = black
15.}
```

Обход в ГЛУБИНУ: три цвета



не пройдена

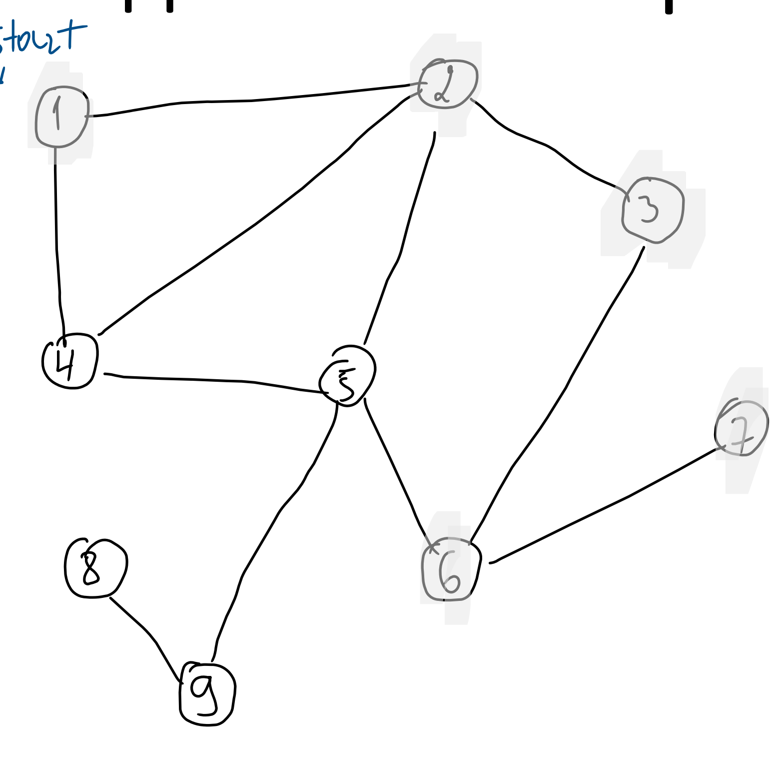
в процессе прохода

пройдена

! Следующая всегда с меньшим номером

```
1.DFS (G) {
2.     For  u из G.V do
3.         u.color = white
4.     For  u из G.V do
5.     If  u.color == white  then
6.             Visit (G, u)
7.}
8.
9.Visit (G) {
10.    u.color = gray
11.    For  v  из G.V[u]  do
12.        If  v.color == white  then
13.            Visit (G, v)
14.    u.color = black
15.}
```

Обход в ГЛУБИНУ: три цвета



не пройдена

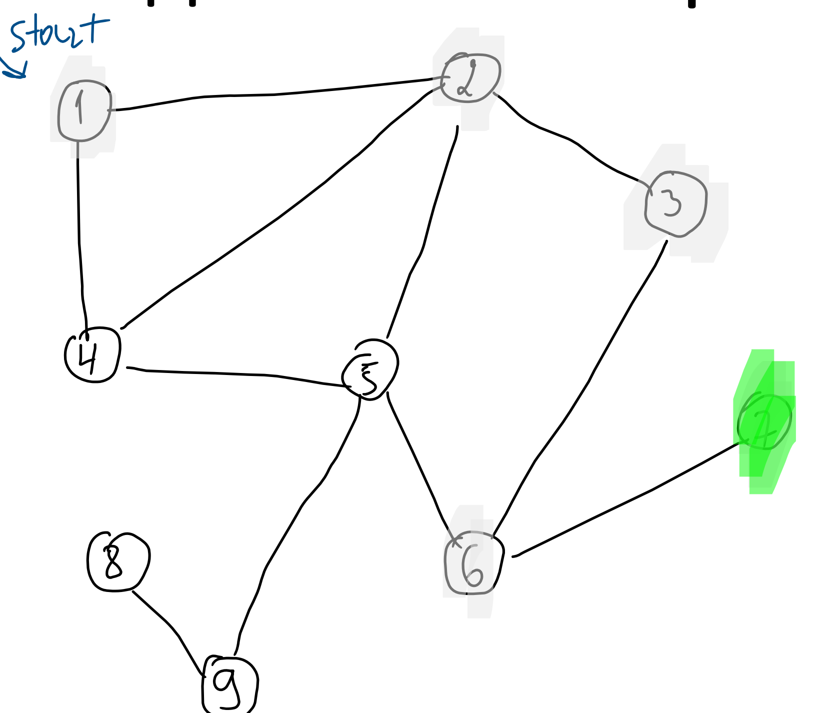
в процессе прохода

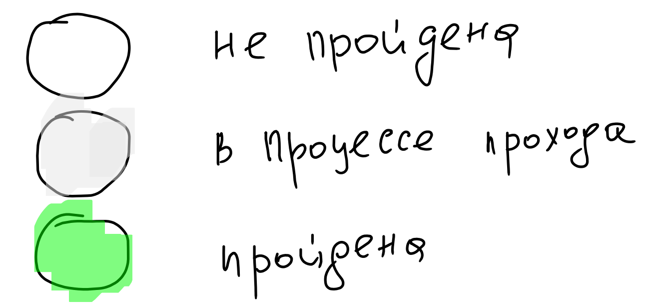
пройдена

! Следующая всегда с меньшим номером

```
1.DFS (G) {
2.     For  u из G.V do
3.         u.color = white
4.     For  u из G.V do
5.     If  u.color == white  then
6.             Visit (G, u)
7.}
8.
9.Visit (G) {
10.    u.color = gray
11.    For  v  из G.V[u]  do
12.        If  v.color == white  then
13.            Visit (G, v)
14.    u.color = black
15.}
```

Обход в ГЛУБИНУ: три цвета

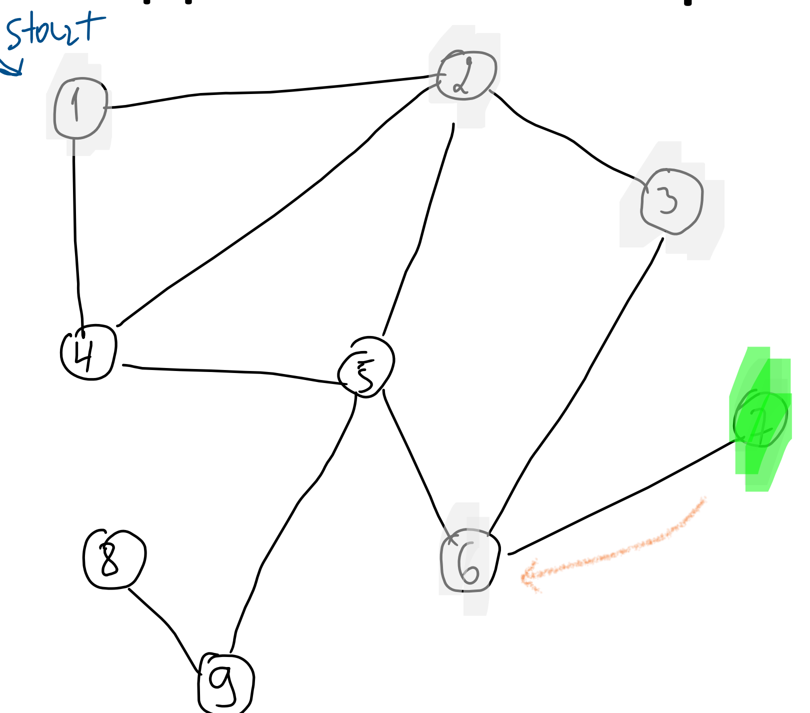


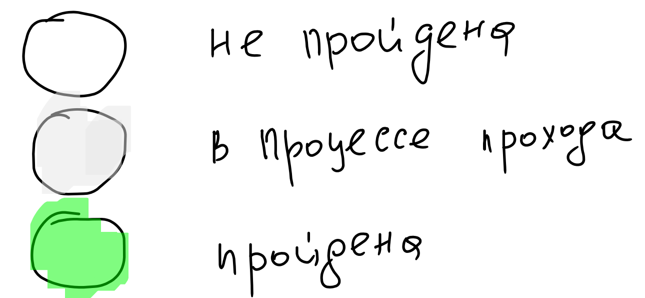


! Следующая всегда с меньшим номером

7

Обход в ГЛУБИНУ: три цвета

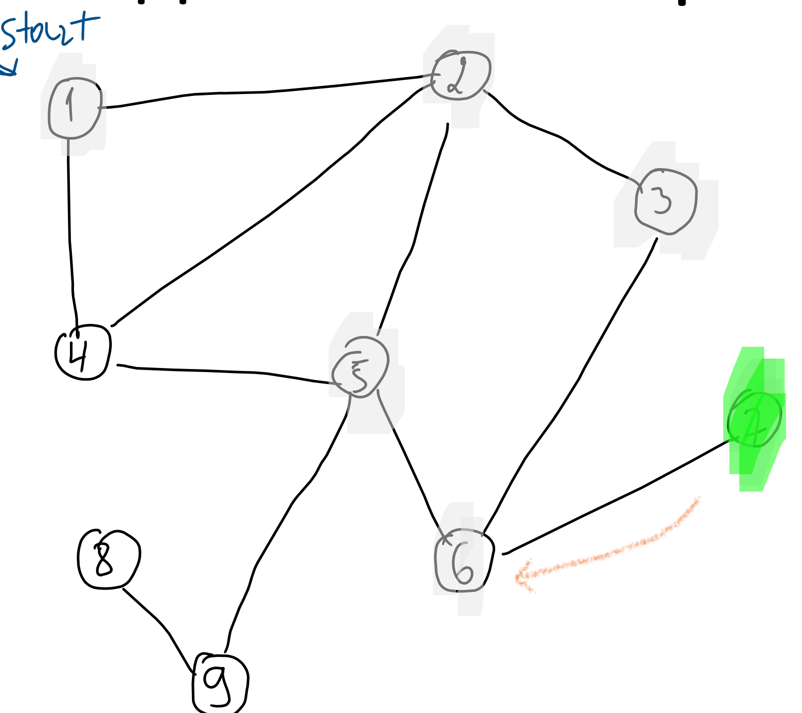


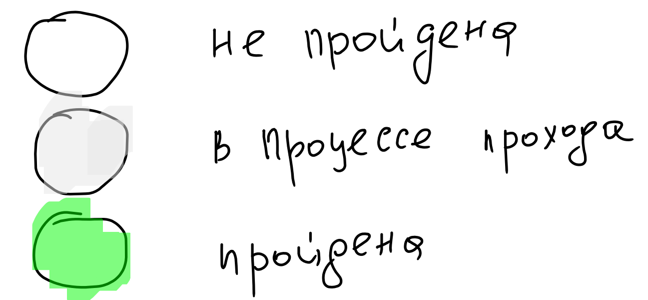


! Следующая всегда с меньшим номером

7

Обход в ГЛУБИНУ: три цвета

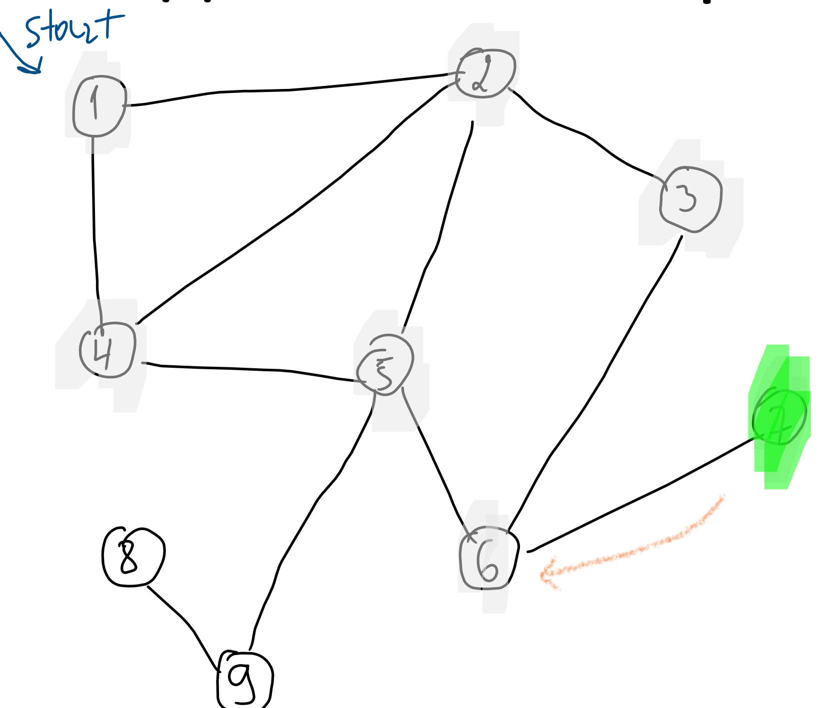


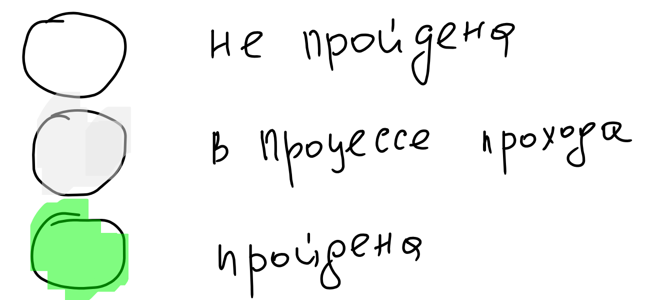


(!) Следующая всегда с меньшим номером

7

Обход в ГЛУБИНУ: три цвета

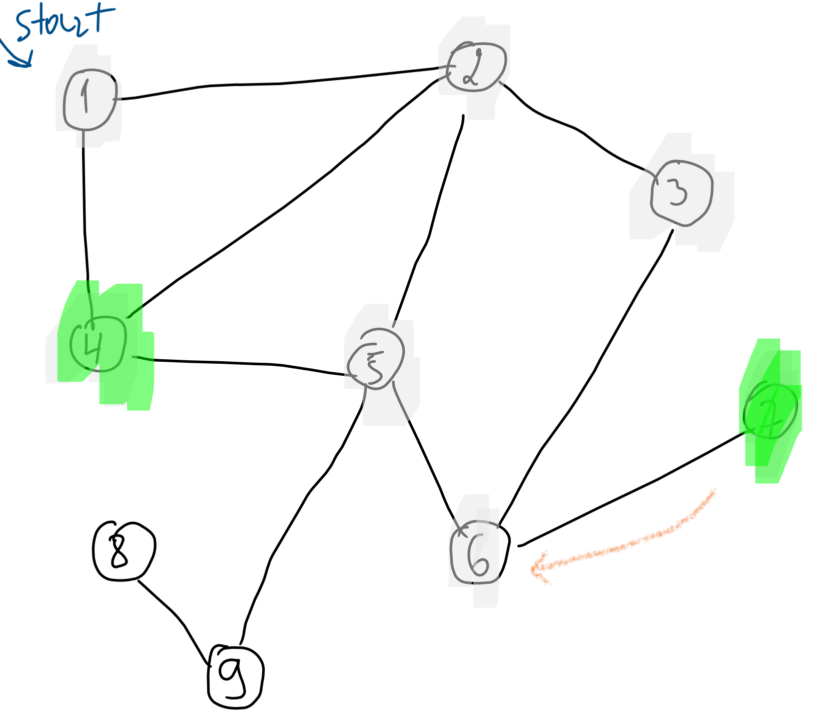


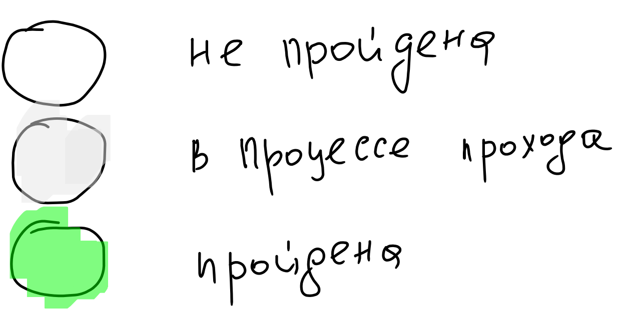


(!) Следующая всегда с меньшим номером

7

Обход в ГЛУБИНУ: три цвета





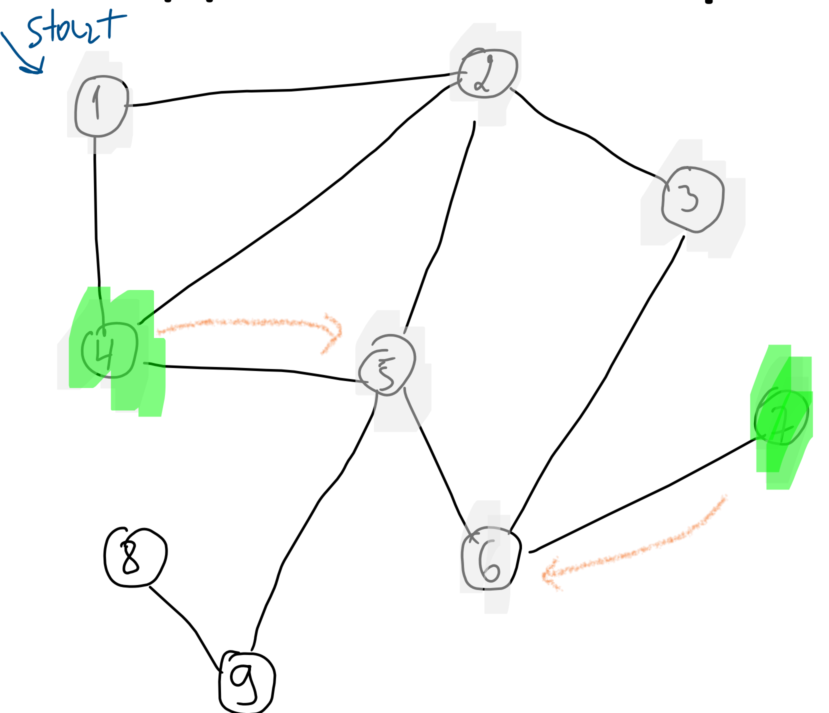
! Следующая всегда с меньшим номером

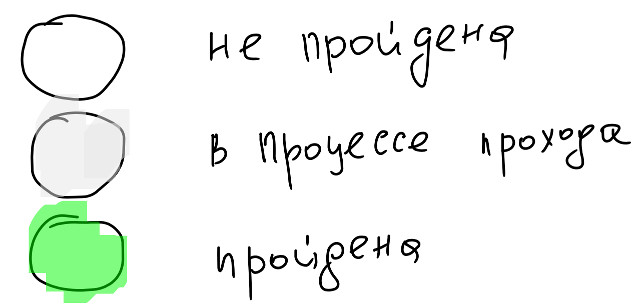
7, 4,

?

серое — серое !

Обход в ГЛУБИНУ: три цвета

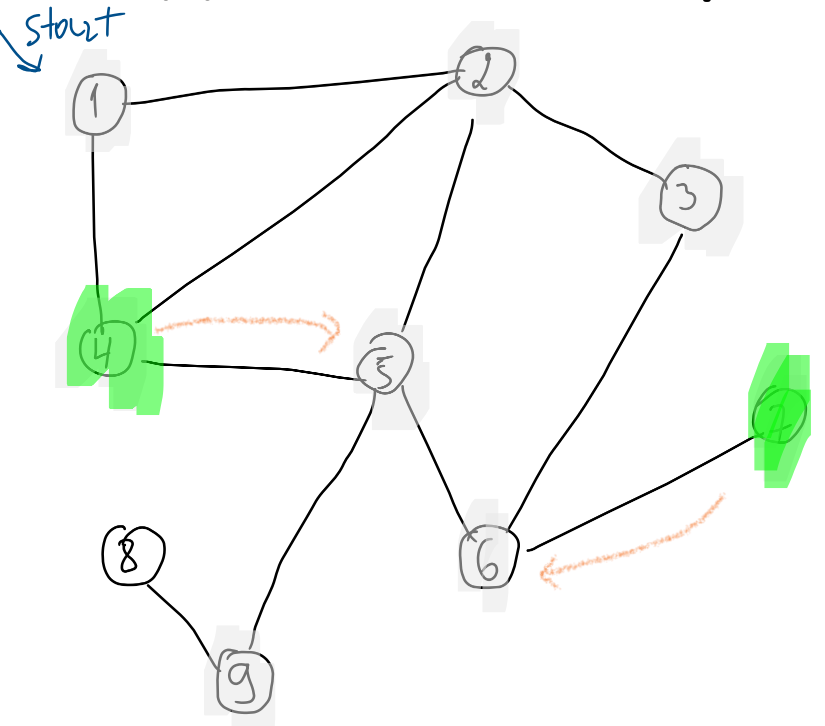


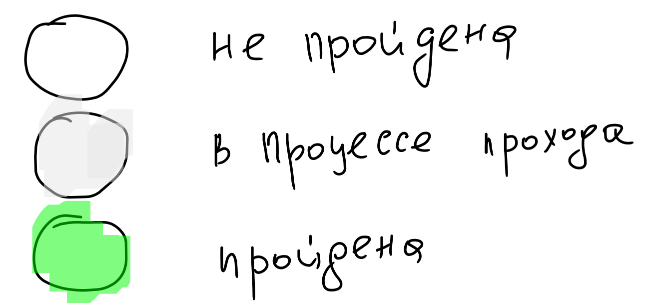


! Следующая всегда с меньшим номером

7, 4,

Обход в ГЛУБИНУ: три цвета

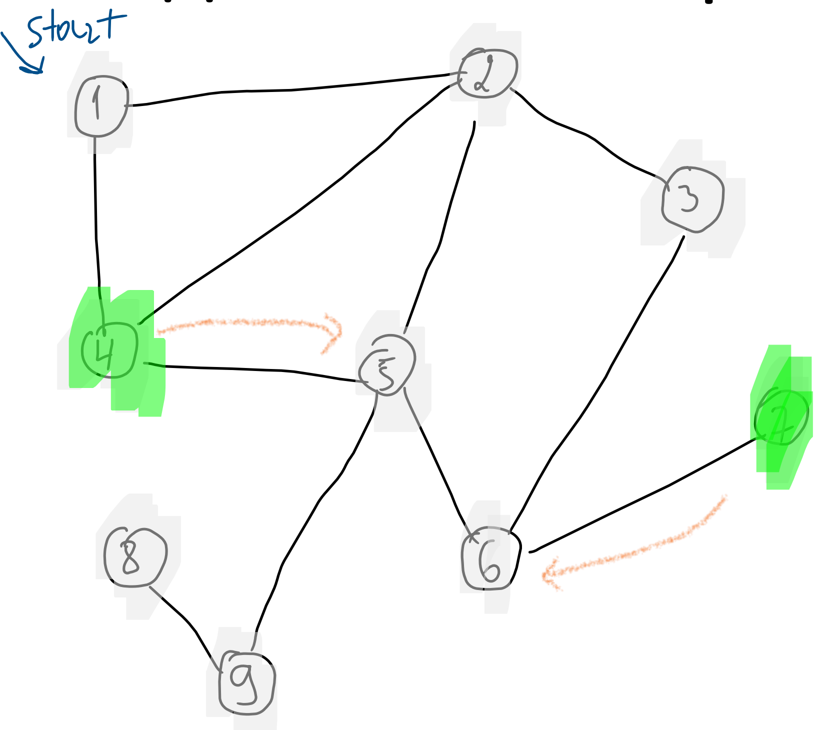


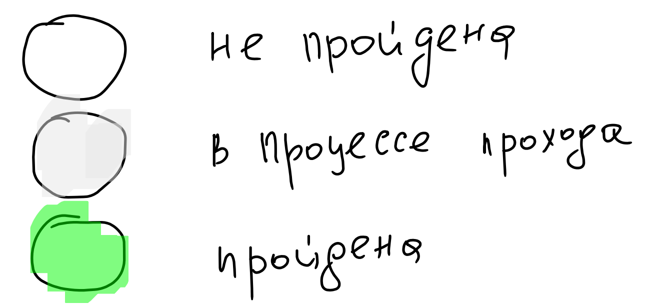


(!) Следующая всегда с меньшим номером

7, 4,

Обход в ГЛУБИНУ: три цвета

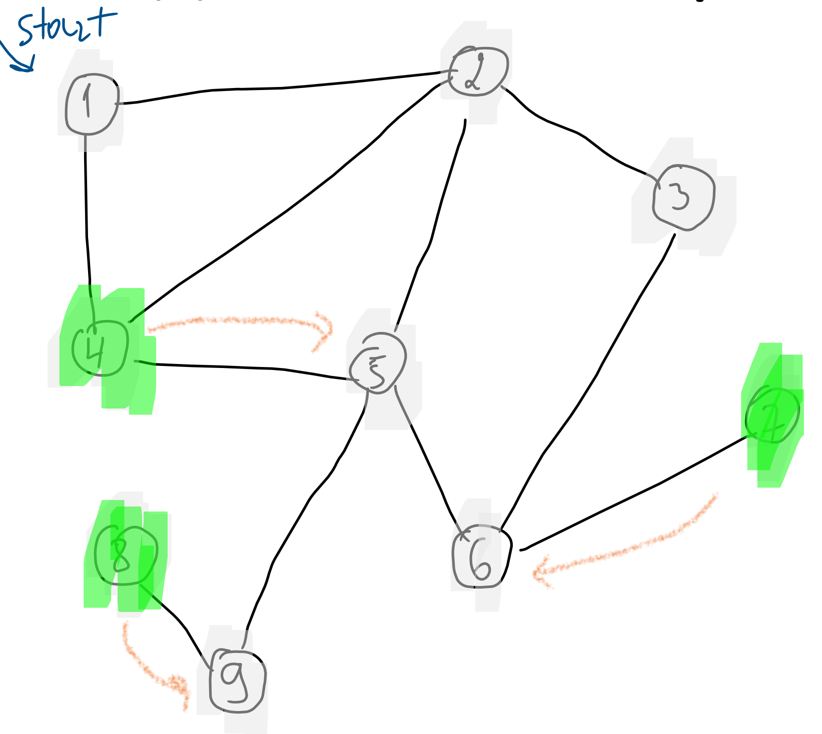


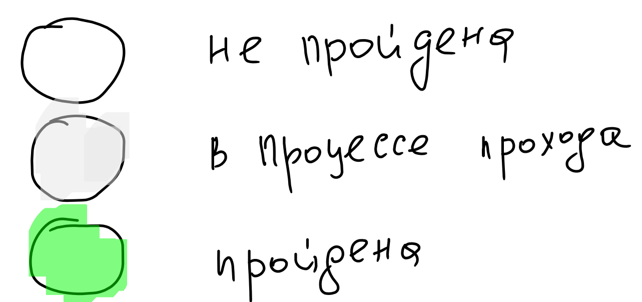


(!) Следующая всегда с меньшим номером

7, 4,

Обход в ГЛУБИНУ: три цвета

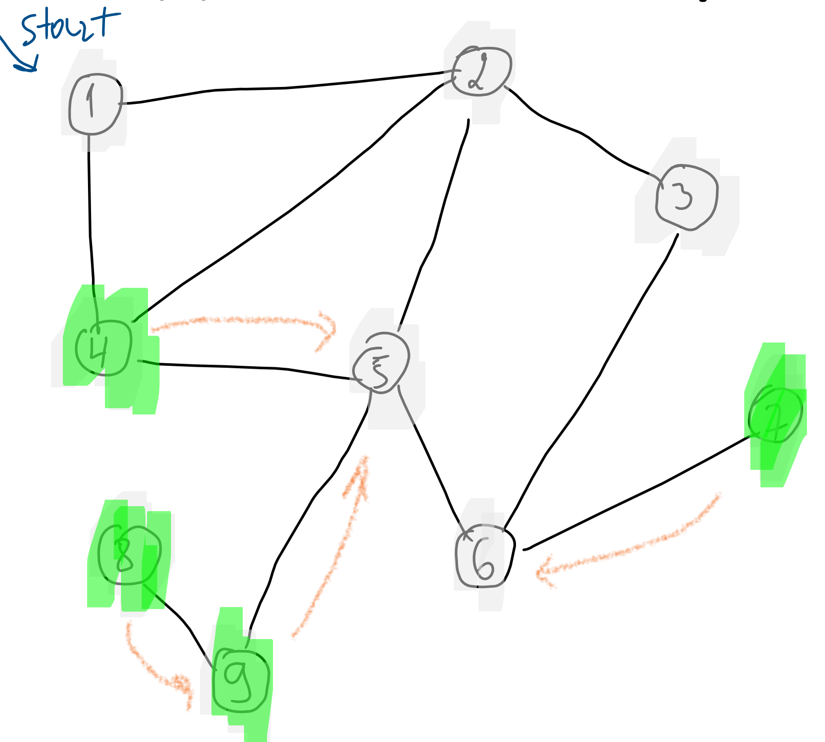


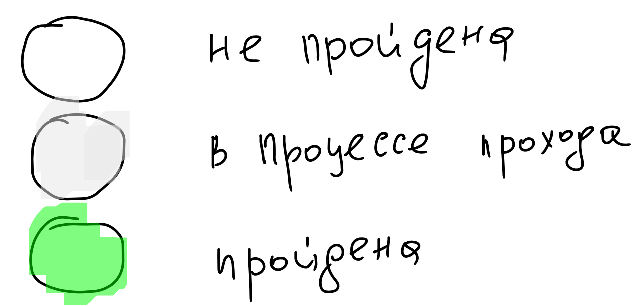


(!) Следующая всегда с меньшим номером

7, 4, 8

Обход в ГЛУБИНУ: три цвета

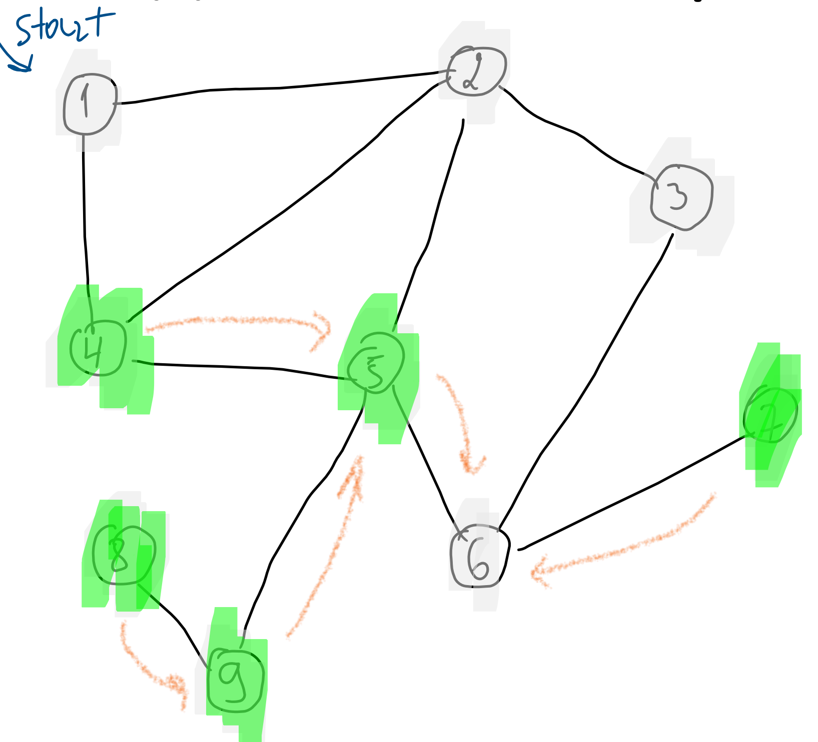


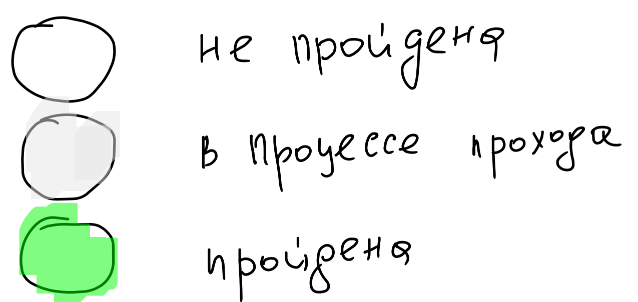


(!) Следующая всегда с меньшим номером

7, 4, 8, 9

Обход в ГЛУБИНУ: три цвета

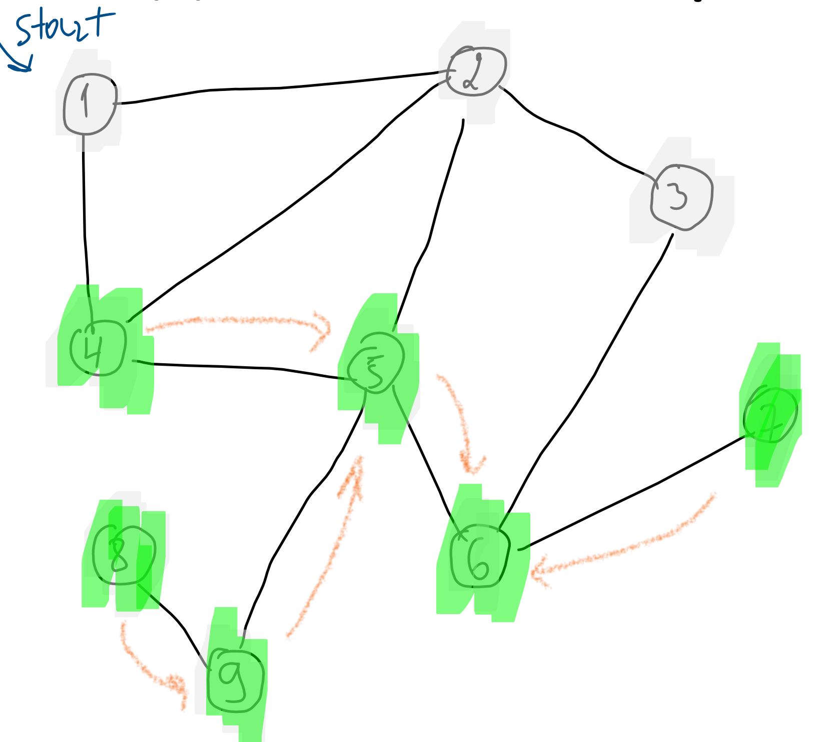


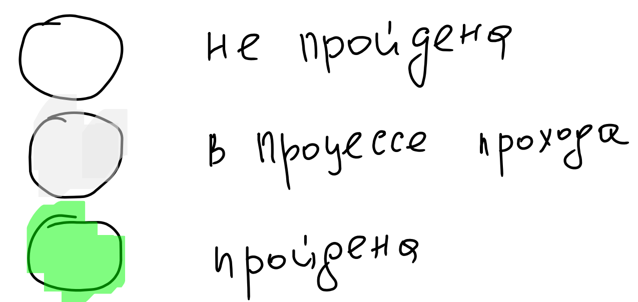


(!) Следующая всегда с меньшим номером

7, 4, 8, 9, 5

Обход в ГЛУБИНУ: три цвета

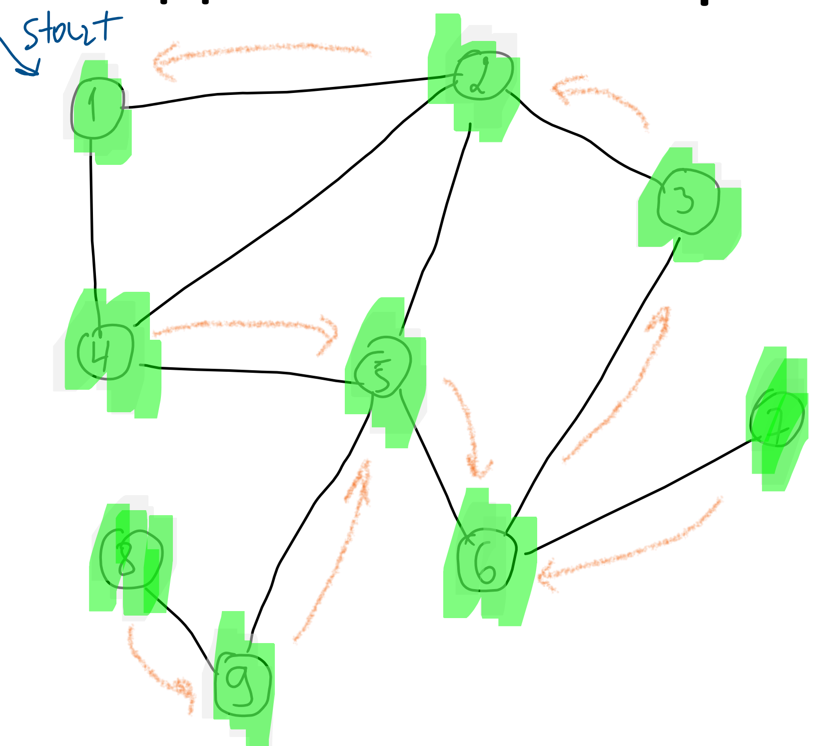


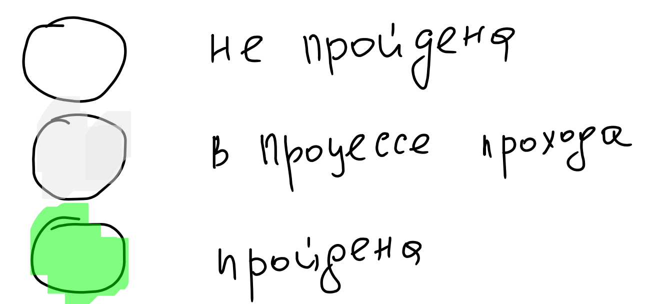


(!) Следующая всегда с меньшим номером

7, 4, 8, 9, 5, 6

Обход в ГЛУБИНУ: три цвета





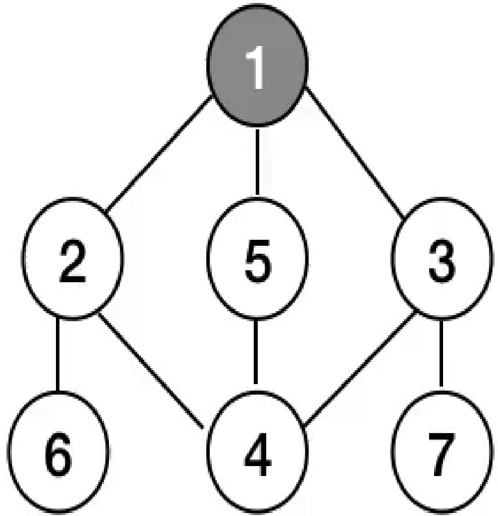
(!) Следующая всегда с меньшим номером

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

Асимптотика: O(|V|+|E|)

Оценим время работы для входного графа G=(V,E), где множество ребер E представлено списком смежности.

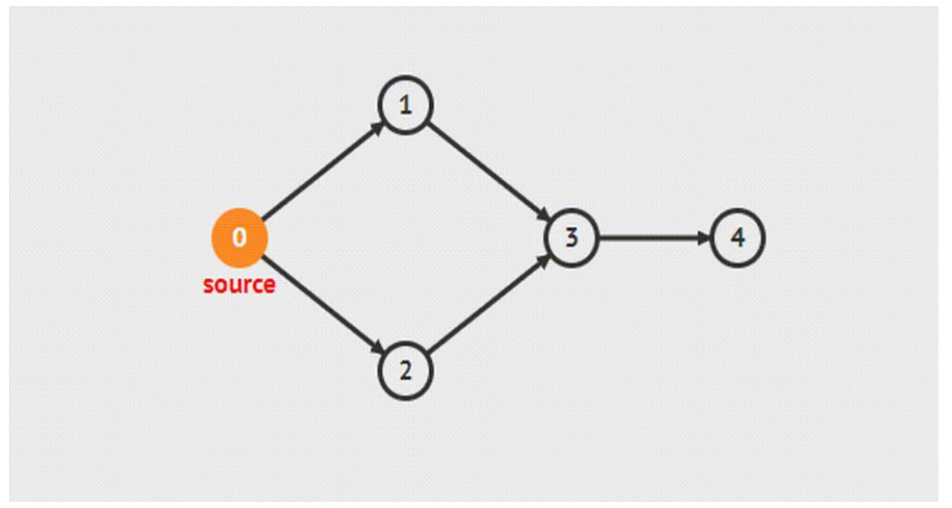
- Просматриваем все вершины.
- Просматриваем все смежные ребра у вершины для завершения обхода или покраски в черный (пройденная).



1

Использование обхода в глубину для поиска цикла

- В случае **ориентированного графа** произведём серию обходов.
- То есть из каждой вершины, в которую мы ещё ни разу не приходили, запустим поиск в глубину, который при входе в вершину будет красить её в серый цвет, а при выходе из нее — в чёрный.
- И, если алгоритм пытается пойти **в серую вершину,** то это означает, что цикл найден.



Использование обхода в глубину для поиска цикла

```
1.// color — массив цветов, изначально все вершины белые
2.func dfs(v: vertex):                  // v — вершина, в которой мы сейчас находимся
3.    color[v] = grey
4.    for (u: vu ∈ E)
5.        if (color[u] == white)
6.            dfs(u)
7.        if (color[u] == grey)
8.            print( «цикл есть» )          // вывод ответа
9.    color[v] = black
```

```
1.DFS (G) {
2.    For  u из G.V do
3.         u.color = white
4.    For  u из G.V do
5.    If  u.color == white  then
6.             Visit (G, u)
7.}
8.
9.Visit (G) {
10.    u.color = gray
11.    For  v  из G.V[u]  do
12.        If  v.color == white  then
13.            Visit (G, v)
14.    u.color = black
15.}
```

Использование обхода в глубину для поиска цикла

```
1.// color — массив цветов, изначально все вершины белые
2.func dfs(v: vertex):                // v — вершина, в которой мы сейчас находимся
3.    color[v] = grey
4.    for (u: vu ∈ E)
5.        if (color[u] == white)
6.            dfs(u)
7.        if (color[u] == grey)
8.            print( «цикл есть» )          // вывод ответа
9.    color[v] = black
```

```
1.DFS (G) {
2.     For  u из G.V do
3.          u.color = white
4.     For  u из G.V do
5.     If  u.color == white  then
6.              Visit (G, u)
7.}
8.
9.Visit (G) {
10.     u.color = gray
11.     For  v  из G.V[u]  do
12.          If  v.color == white  then
13.              Visit (G, v)
14.     u.color = black
15.}
```

Как восстановить весь цикл?

Использование обхода в глубину для поиска цикла

```
1.// color — массив цветов, изначально все вершины белые
2.func dfs(v: vertex):                // v — вершина, в которой мы сейчас находимся
3.    color[v] = grey
4.    for (u: vu ∈ E)
5.        if (color[u] == white)
6.            dfs(u)
7.        if (color[u] == grey)
8.            print( «цикл есть» )         // вывод ответа
9.    color[v] = black
```

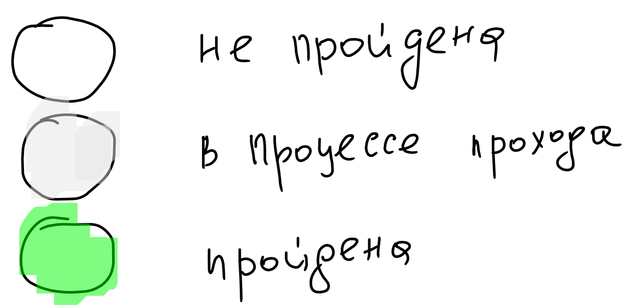
Как восстановить весь цикл? ⇒

массив предков

parent[] = null

и заполняем по мере прохода!

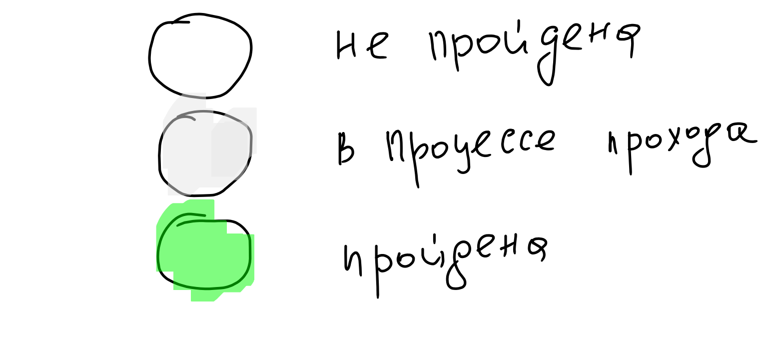
Обход в ГЛУБИНУ: три цвета



start

! Следующая всегда с меньшим номером

серая — серая ⇒ цикл



p[1] = null

p[2] = 1

p[3] = 2

p[4] = 5

p[5] = 6

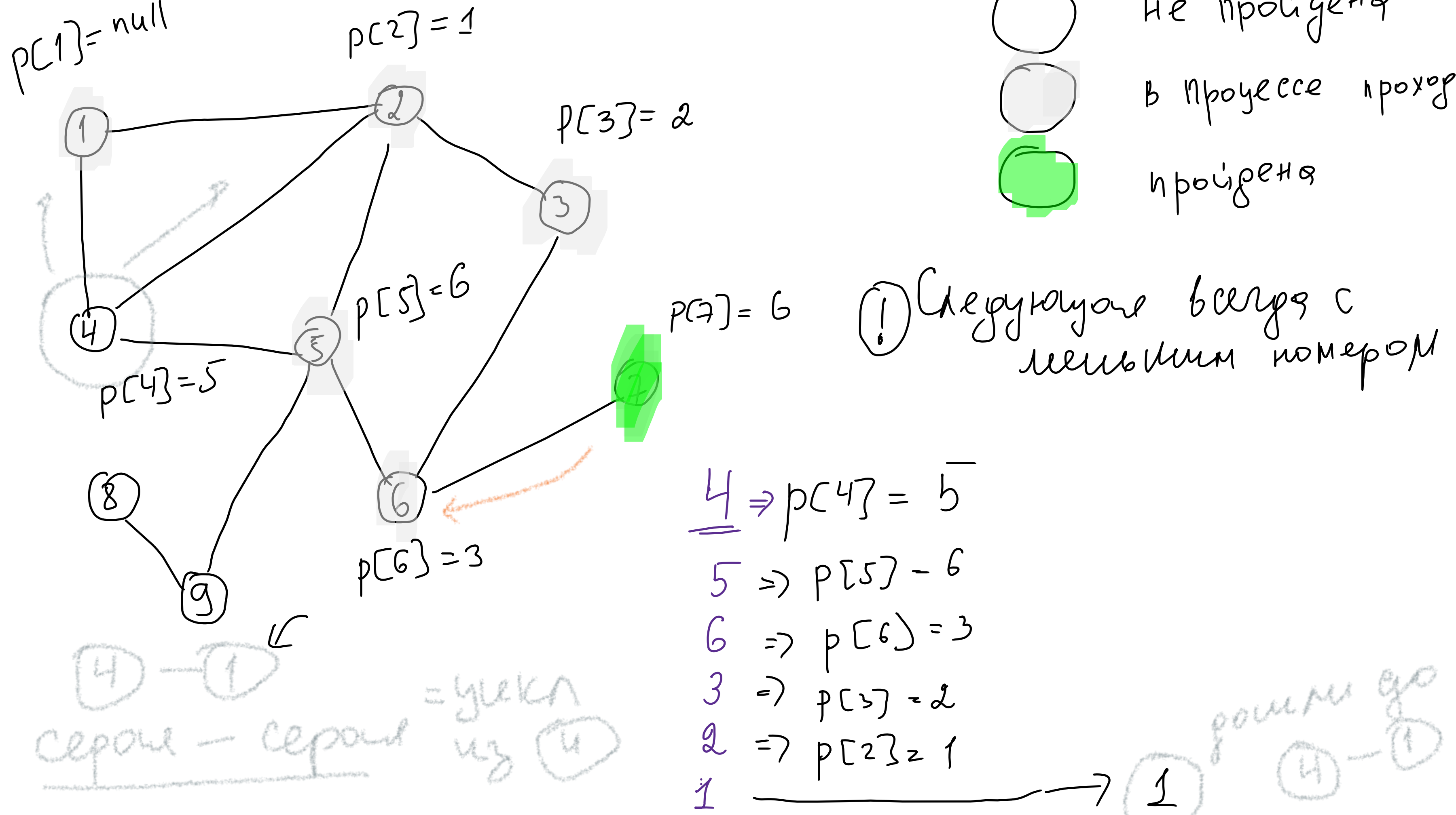
p[6] = 3

p[7] = 6

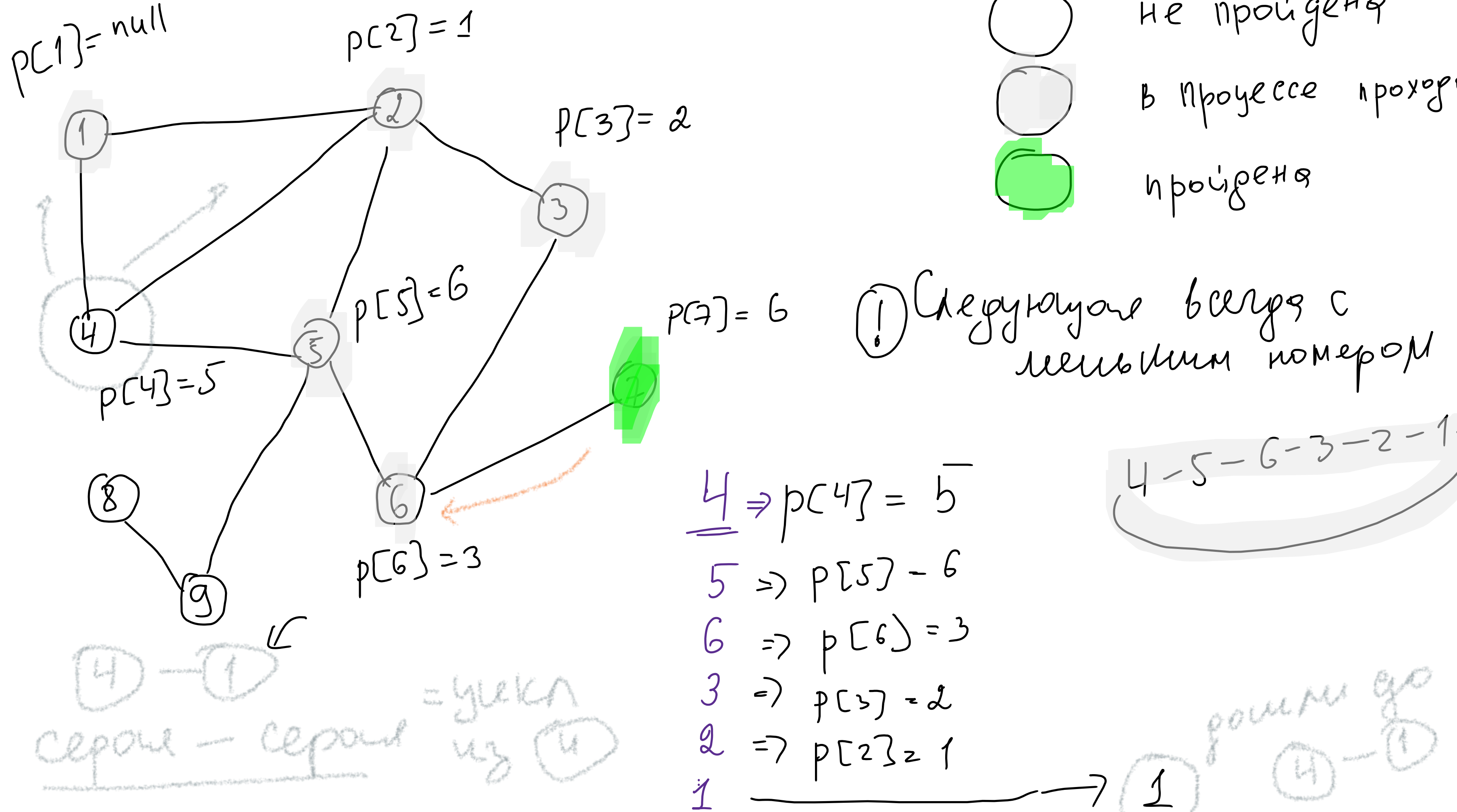
(!) Следующая всегда с меньшим номером

4 — 1 = цикл из 4

серая — серая



p[1]= null
p[2] = 1
p[3]= 2
p[5]=6
p[7]= 6
p[4]=5
p[6] =3
1
2
3
4
5
6
7
8
9
не пройдена
в процессе прохода
пройдена
(!) Следующая всегда с меньшим номером
4 ⇒ p[4] = 5
5 => p[5] - 6
6 => p[6) = 3
3 => p[3] = 2
2 => p[2] = 1
1 → 1
4 — 1
серая — серая = цикл из 4



p[1]= null
p[2] = 1
p[3]= 2
p[5]=6
p[7]= 6
p[4]=5
p[6] =3
не пройдена
в процессе прохода
пройдена
! Следующая всегда с меньшим номером
4-5-6-3-2-1
4 => p[4] = 5
5 => p[5] - 6
6 => p[6] = 3
3 => p[3] = 2
2 => p[2]= 1
1
1
4 — 1
серая — серая = цикл
из 4
дошли до
4 — 1

Использование обхода в глубину для поиска цикла

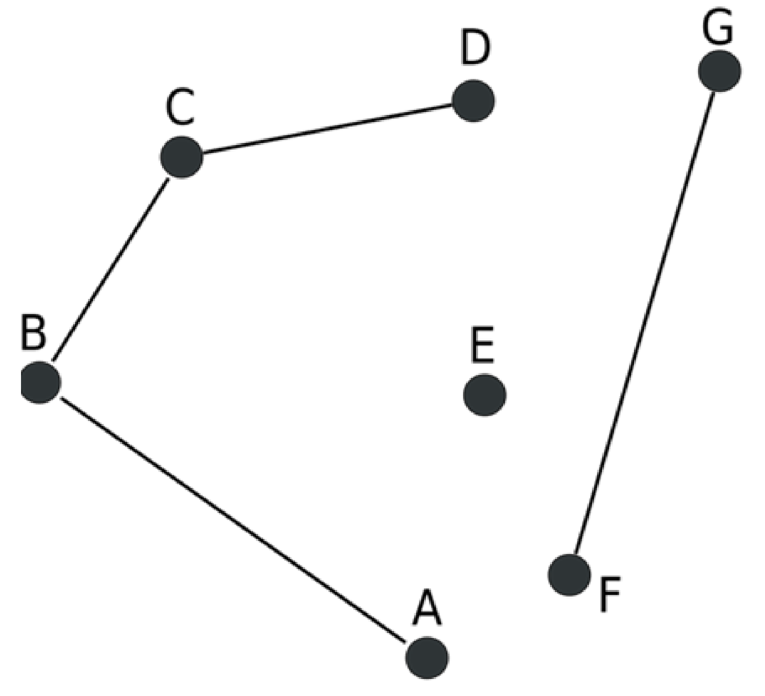
- Как найти другие циклы?
- Как найти цикл в цикле?
- Как найти все возможные

  циклы ?

Поиск компонент связности графа

Алгоритм:

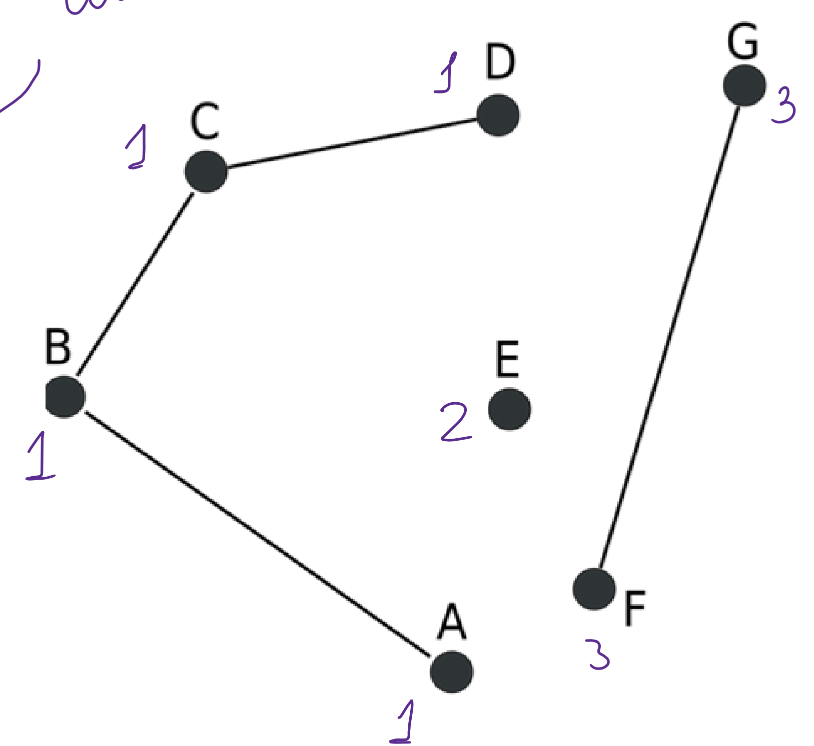
- 1. Помечаем все вершины как не пройденные
- 2. Цикл пока есть не пройденные вершины
    - a. Запускаем обход в глубину от вершины
        - i. Все пройденные вершины собираем в первую компоненту
    - b. Ищем не пройденную вершину
- 3. Выводим все компоненты графа



Поиск компонент связности графа

Алгоритм:

- 1. Помечаем все вершины как не пройденные
- 2. Цикл пока есть не пройденные вершины
    - a. Запускаем обход в глубину от вершины
        - i. Все пройденные вершины собираем в первую компоненту
    - b. Ищем не пройденную вершину
- 3. Выводим все компоненты графа



Топологическая сортировка

- Только для ациклических ориентированных графов!
- Сортирует граф, таким образом, что для любой дуги (u, v) u будет перед v

- **<u>Алгоритм</u>**
- Топологическая сортировка (G(V, E))
  - Обход графа G(V, E) в глубину
    - Каждую пройденную (черную) вершину помещаем в стек
  - Достать все вершины из стека

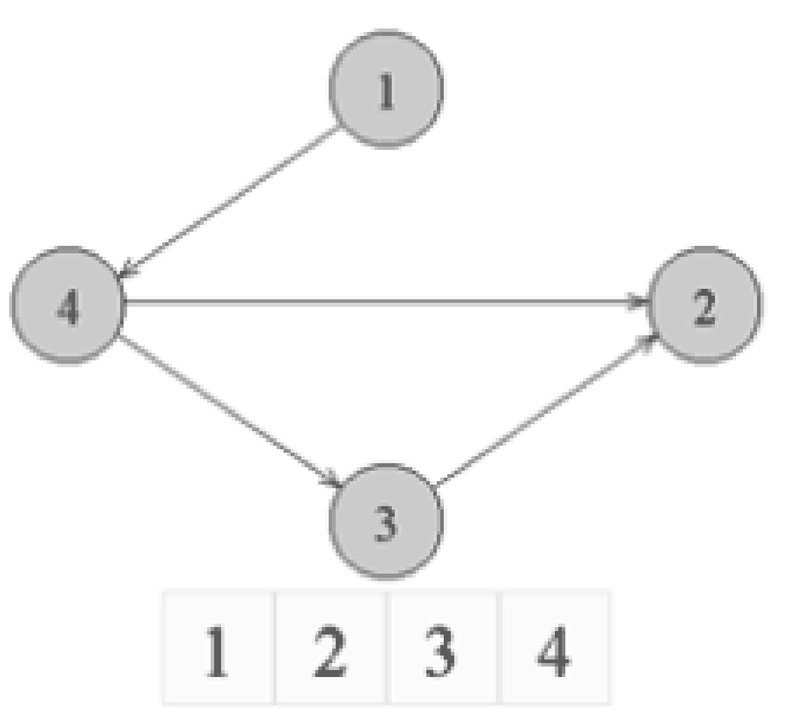
- *С помощью топологической сортировки можно найти гамильтонов путь!*

Топологическая сортировка

- Топологическая сортировка (G(V, E))
  - Обход графа G(V, E) в глубину
    - Каждую пройденную (черную) вершину помещаем в стек
  - Достать все вершины из стека

```
1.function topologicalSort():
2.    проверить граф G на ацикличность
3.    fill(visited,false)
4.    for v∈V(G)
5.        if not visited[v]
6.             dfs(v)
7.    ans.reverse()
8.
9.function dfs(u):
10.    visited[u]=true
11.    for uv∈E(G)
12.        if not visited[v]
13.            dfs(v)
14.    ans.pushBack(u)
```

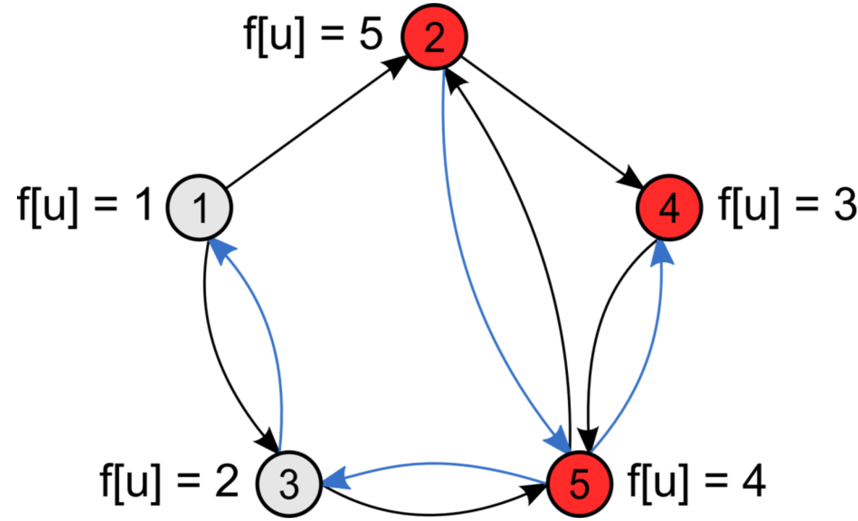
Топологическая сортировка




```
1.function topologicalSort():
2.     проверить граф G на ацикличность
3.     fill(visited,false)
4.     for v∈V(G)
5.         if not visited[v]
6.              dfs(v)
7.     ans.reverse()
8.
9.function dfs(u):
10.     visited[u]=true
11.     for uv∈E(G)
12.         if not visited[v]
13.             dfs(v)
14.     ans.pushBack(u)
```

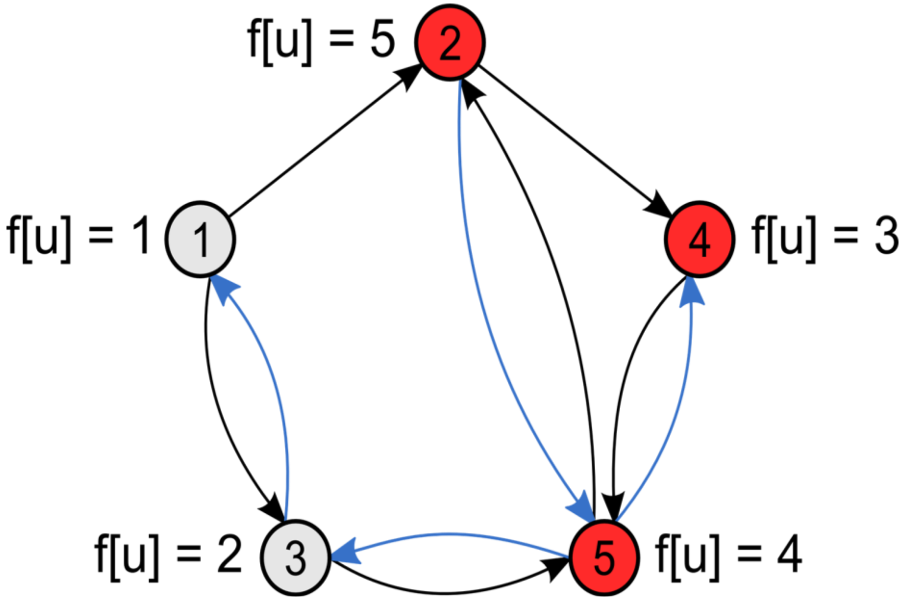
Конденсация: компоненты сильной связности

- Компоненты сильной связности в графе G можно найти с помощью поиска в глубину в 3 этапа:
  - 1. Построить граф H с обратными (инвертированными) рёбрами
  - 2. Выполнить в H поиск в глубину и найти f[u] — время окончания обработки вершины u
  - 3. Выполнить поиск в глубину в G, перебирая вершины во внешнем цикле в порядке убывания f[u]
- Полученные на 3-ем этапе деревья поиска в глубину будут являться компонентами сильной связности графа G.
- Так как компоненты сильной связности G и H графа совпадают, то первый поиск в глубину для нахождения f[u] можно выполнить на графе G, а второй — на H.



Вершины 2, 4, 5 сильносвязаны. Синим цветом обозначен обод DFS по инвертированным ребрам

Конденсация: компоненты сильной связности



Вершины 2, 4, 5 сильносвязаны.
Синим цветом обозначен обод DFS по инвертированным ребрам

```
1.function dfsG(u):
2.        for v in G.V[u]
3.            if not v.visited
4.                dfsG(v)
5.        u. visited = true
6.     stack. push (u)
7.
8.    function dfsH(u):
9.        component[u] = count
10.        for v in H.V[u]
11.            if component[v]==0
12.                dfsH(v)
13.
14.    function main():
15.        формируем графы  G и H
16.     обнулям массив component
17.        for u in V
18.            if not u.visited
19.                dfsG(u)
20.        count = 1
21.        for u = stack.pop
22.            if component[u]==0
23.                dfsH(u)
24.                count ++
```

Спасибо за внимание!

www.ifmo.ru

ITsMOre than a UNIVERSITY