Белорусский государственный технологический университет

Кафедра Информационных Систем и Технологий

**Курс «Математическое программирование»**

**Отчёт по лабораторной работе №5**

**Оптимизационные алгоритмы на графах**

**Вариант 12**

Выполнила: Смелова В.В.

ФИТ 2 курс 4 группа

Проверила: Харланович А.В.

Минск 2020

**ЦЕЛЬ РАБОТЫ:** освоить сущность и программную реализацию: а) способов представления графов; б) алгоритмов поиска в ширину и глубину; в) алгоритма топологической сортировки графов.

**Задание 1.**  Ориентированный граф **G** взять в соответствии с вариантом. Осуществить алгоритмы поиска в ширину и глубину, а также алгоритма топологической сортировки аналогично примерам, рассмотренным на алгоритмах.

1. **Сортировка в ширину**

|  |  |
| --- | --- |
| Вариант 12 |  |

Текущее состояние алгоритма хранится в следующих структурах памяти:

**Q** – очередь вершин,

**C** – массив окраски вершин,

**D** – массив расстояний,

**P** – массив предшествующих вершин.

По условию, граф имеет 5 вершин, пронумерованных начиная с нуля. В качестве стартовой вершины выбрана вершина с номером 0.

Шаг 1. В очереди – нулевая вершина, окрашенная в серый, остальные вершины белые. Расстояние до нее равно 0, предыдущие вершины не определены.

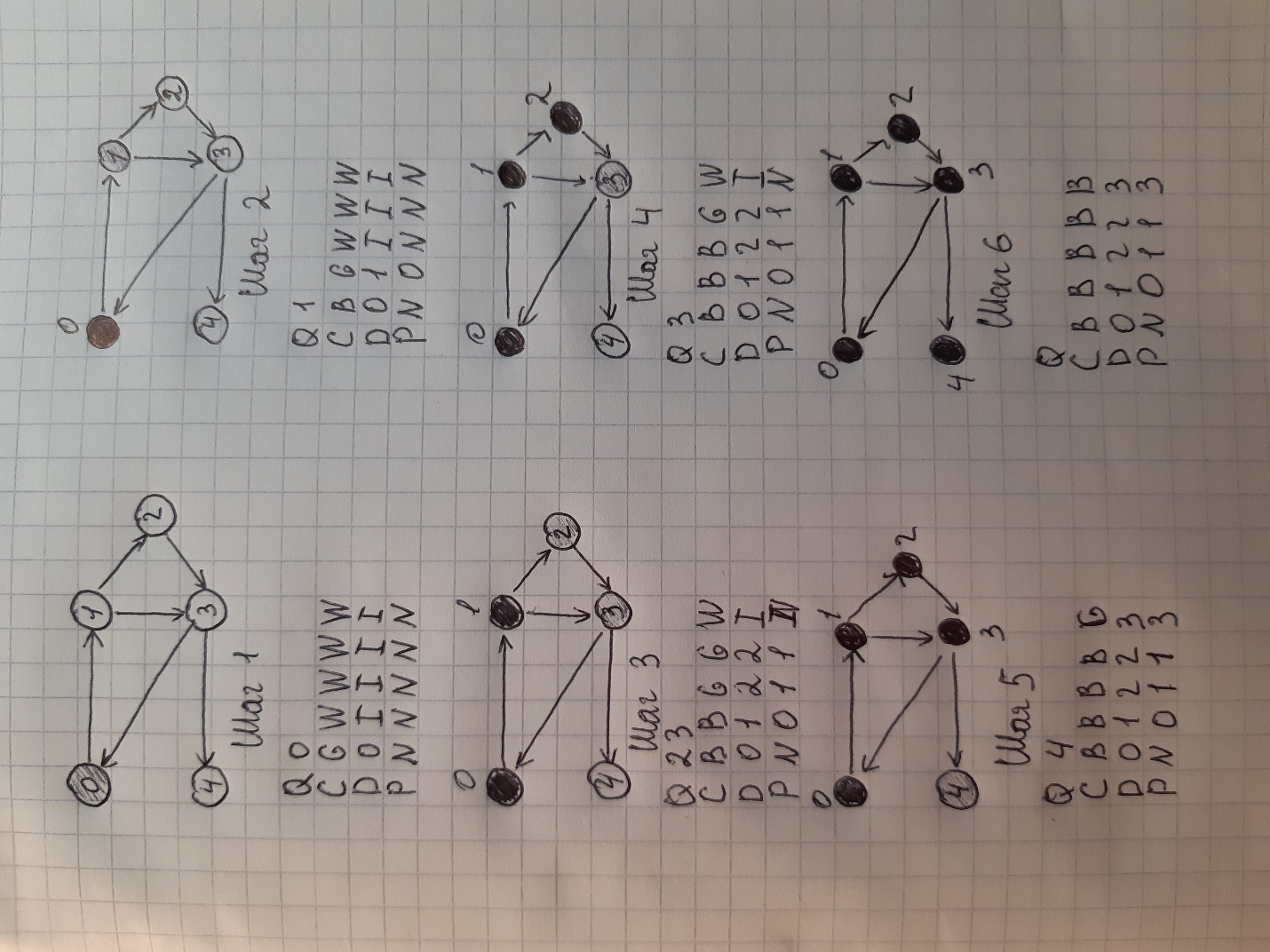
Шаг 2. Сдвигаемся с 0 на 1, закрашиваем 0 в черный, а 1 – в серый. В очереди первая вершина.

Шаг 3. Ищем смежные вершины для вершины 1. Вершины 2 и 3 окрашиваем в серый, вершину 1 – в черный, помещаем в очередь вершины 2 и 3.

Шаг 4. Берем из очереди вершину 2 и ищем смежные вершины. Кроме помеченной вершины 3 таких не имеется, следовательно вершина 2 окрашивается в черный, и мы переходим с вершине, следующей по очереди, то есть к вершине 3.

Шаг 5. Ищем смежные вершины для вершины 3. Из непомеченных имеется лишь 4. Помещаем ее в очередь и окрашиваем в серый. Вершина 3 окрашивается в черный.

Шаг 6. Для вершины 4 нет переходов, очередь пуста, следовательно, вершина 4 окрашивается в черный, и алгоритм завершает работу.



На выходе алгоритма мы имеем массив P, показывающий порядок обхода. Так для каждой вершины можно определить кратчайший путь от корня.

1. **Сортировка в глубину**

Текущее состояние алгоритма хранится в следующих структурах памяти:

**C** – массив окраски вершин,

**D** – время окраски вершин в серый цвет,

**P** – массив предшествующих вершин,

**F** – время окраски в чёрный цвет.

Кроме того, используется переменная t, текущее значение которой – номер шага алгоритма.

По условию, граф имеет 5 вершин, пронумерованных начиная с нуля. В качестве стартовой вершины выбрана вершина с номером 0.

Шаг 1. Стартовая вершина закрашена в серый.

Шаг 2. Спускаемся по единственной выходящей из вершины дуге в вершину 1. Окрашиваем ее в серый.

Шаг 3. У вершины 1 две смежных вершины – 2 и 3, спускаемся сначала во вторую – окрашиваем ее в серый.

Шаг 4. У вершины 2 всего одна смежная вершина – вершина 3. Окрашиваем ее в серый.

Шаг 5. У вершины 3 всего одна смежная – 4. Окрашиваем ее в серый.

Шаг 6. Из вершины 4 не выходят дуги, следовательно мы достигли глубины. Окрашиваем вершину 4 в черный. Начинаем подъем.

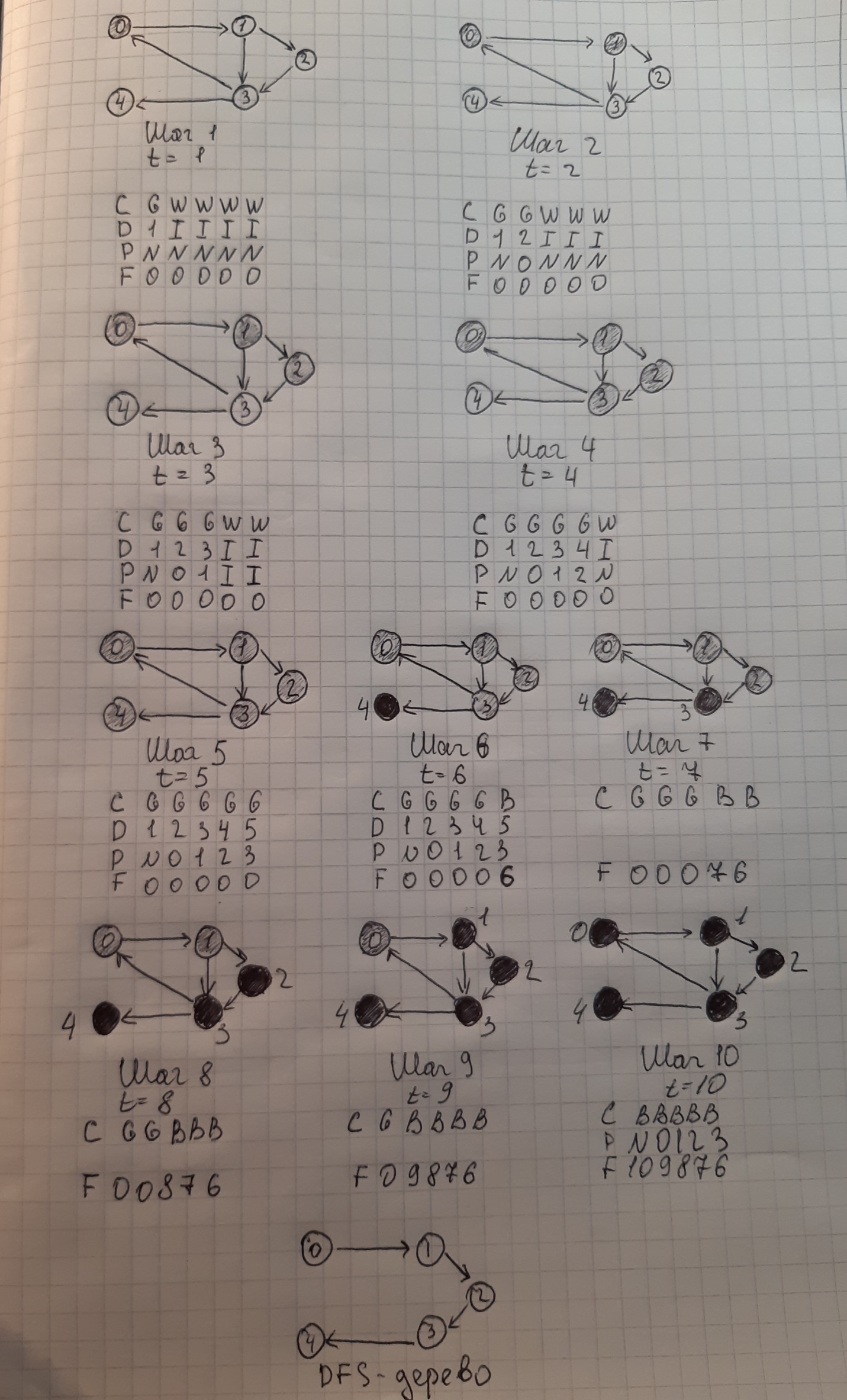
Шаг 7. Возвращаемся в вершину 3. Больше смежных непройденных(!) вершин у нее нет, следовательно окрашиваем ее в черный.

Шаг 8. В вершину 3 мы пришли из вершины 2, возвращаемся к ней. У нее не осталось непройденных смежных ребер, следовательно окрашиваем ее в черный.

Шаг 9. В вершину 2 пришли из 1, возвращаемся. Нет больше смежных вершин – вершину 1 в черный цвет.

Шаг 10. Возвращаемся в стартовую вершину и окрашиваем ее в черный.

На выходе имеем массив N, показывающий порядок обхода вершин графа.

****

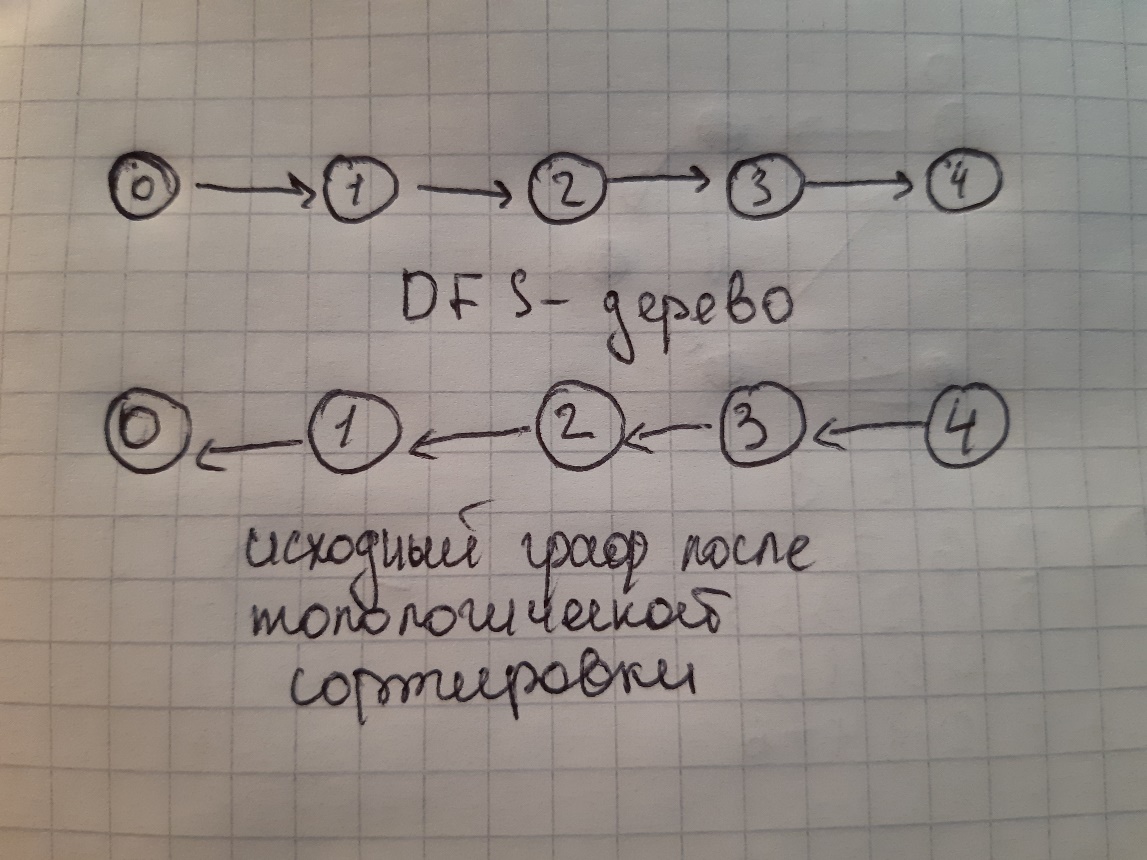
1. Топологическая сортировка

Алгоритм топологической сортировки:

Получить последовательность вершин исходного графа при обходе его с помощью алгоритма сортировки в глубину.

Построить новую последовательность вершин, расположив вершины графа в обратном порядке.

Если требуется, перенумеровать вершины.



**Задание 2.** Осуществить программную реализацию алгоритмов на C++. Разработать структуры **AMatrix** и **АList**  для представления ориентированного графа матричным и списковым способом. Разработать функции преобразования из одного способа представления в другой. Разработать функцию **BFS** обхода вершин графа, используя метод поиска в ширину. Продемонстрировать работу функции. Копии экрана вставить в отчет.

Для обозначения алгоритма поиска в ширину обычно используют аббревиатуру ***BFS*** (***Breadth-first search***). Алгоритм подразумевает, что задана исходная (***стартовой***) вершина, и основывается на простом правиле: при выборе очередной вершины предпочтение отдается ближайшей. При этом считается, что все дуги графа имеют единичную длину. Сначала посещается стартовая вершина, затем все вершины, смежные ей (т. е. находящиеся на расстоянии 1), после чего вершины, находящиеся на расстоянии 2 от стартовой и т.д.

Очевидно, что алгоритм BFS посетит только те вершины, для которых существует последовательность дуг, связывающая с ними стартовую вершину. Обычно, по окончании работы алгоритма, осуществляется проверка на полноту обхода и, если имеются вершины, которые не посещались, то среди них выбирается любая, назначается стартовой и алгоритм выполняется снова.

Текущее состояние алгоритма хранится в следующих структурах памяти: **Q** – очередь вершин, **С** – массив окраски вершин, **D** – массив расстояний и **P** – массив предшествующих вершин. Все массивы имеют размерность, равную количеству вершин в графе.

Очередь **Q** (структура памяти, реализующая алгоритм «первый вошел − первый вышел») используется для промежуточного хранения номеров вершин. На каждом шаге алгоритма, кроме может быть нескольких последних, в очередь помещаются номера вершин в порядке их обнаружения. На каждом шаге, кроме первого, из очереди извлекается очередной номер вершины, подлежащей отметке о посещении (фиксации). На первом шаге алгоритма в очередь помещается номер стартовой вершины. На последнем шаге очередь пуста.

Массив **C** используется для хранения состояния вершин. С каждым из трех возможных состояний обычно связывают цвет: белый (**W**) – вершина не посещалась, серый (**G**) – вершина посещалась, черный (**B**) – фиксирован факт посещения вершины. На первом шаге алгоритма стартовая вершина окрашивается в серый цвет, а остальные – в белый. На последнем шаге все вершины становятся черными.

В массиве **D** для каждой вершины хранятся расстояния от стартовой вершины. На первом шаге для стартовой вершины в массиве **D** устанавливается значение 0, а для остальных вершин – значение «бесконечность» (**I**). На последнем шаге алгоритма для всех доступных вершин будут заполнены значения, равные их расстоянию от стартовой вершины.

Массив **P** позволяет восстановить порядок обхода вершин и хранит для каждой вершины, кроме стартовой, предшествующую в обходе вершину. На первом шаге алгоритма всем элементам массива присваивается значение, интерпретируемое как «пустота» (**N**). На последнем шаге алгоритма для всех доступных вершин будут заполнены значения, равные номеру предшествующей вершины в порядке обхода.

Полученный в результате работы массив **P** позволяет построить так называемое BFS-дерево.

***BFS-дерево*** – это дерево, множество вершин которого является подмножеством вершин исходного графа, связанных дугами в порядке их посещения (в соответствии с массивом **P**), а корнем – стартовая вершина.

Алгоритм BFS сводится к следующей последовательности шагов.

1. Инициализировать массивы **С**, **D**, **P**. Стартовую вершину **s** поместить в очередь **Q**. и окрасить в серый цвет: **C[s] = G**. Для стартовой вершины установить расстояние, равное нулю: **D[s] = 0**.
2. Если очередь **Q** пуста, то работа алгоритма завершена, в противном случае перейти к следующему шагу.
3. Выбрать из очереди **Q** вершину **k** и окрасить ее в черный цвет: **С[k] = B**.
4. Построить множества **J** вершин белого цвета смежных вершине **k**. Если таких вершин нет, то перейти к шагу 2, иначе – к следующему шагу.
5. Каждую вершину **j** из множества **J** поместить в очередь **Q**. Обычно (но не обязательно) в очередь вершины помещаются в порядке возрастания номеров.
6. Каждую вершину **j** из множества **J** окрасить в серый цвет: **С[j] = G**.
7. Для каждой вершины **j** из множества **J** вычислитьрасстояние: **D[j] = D[k] + 1**.
8. Для каждой вершины **j** из множества **J** указать предшествующую вершину: **P[j] = k**.
9. Перейти к шагу 3.

Полученный в результате работы массив **P** позволяет построить так называемое BFS-дерево.

**Задание 3.**  Разработать функцию **DFS** обхода вершин графа, используя метод поиска глубину. Продемонстрировать работу функции. Копии экрана вставить в отчет.

Для обозначения алгоритма поиска в глубину используют аббревиатуру ***DFS*** (***Depth-first search***). Как и для поиска в ширину, задается стартовая вершина. Алгоритм описывается следующим образом: для каждой не пройденной вершины, начиная со стартовой, необходимо найти все смежные вершины и повторить поиск для каждой.

Назначение и размерность массивов **С** (массив окраски вершин) и **P** (массив предшествующих вершин) такие же, как и в алгоритме BFS. В массиве **D** для каждой вершины записывается время обнаружения (шаг окраски в серый цвет). Массив **F** предназначен для хранения времени фиксации (шага окраски в черный цвет) вершины. Кроме того, используется переменная **t**, текущее значение которой – номер шага алгоритма.

В основе алгоритма DFS лежит рекурсивная процедура **Visit**, имеющая один входной параметр **k** – вершину графа.

Опишем пошагово процедуру **Visit**.

1. Принять параметр **k** – вершину графа.
2. Вершину **k**  окрасить в серый цвет: **C[k] = G**.
3. Увеличить номер шага: **t = t + 1**.
4. Подсчитать расстояние до вершины: **D[k] = t**. Расстояние до вершины в алгоритме DFS совпадает с номером шага, на котором эта вершина была обнаружена (окрашена в серый цвет).
5. Построить множества **J** вершин белого цвета, смежных вершине **k**. Если таких вершин нет, то перейти к шагу 8.
6. Для каждой вершины **j** из множества **J** указать предшествующую вершину: **P[j] = k**.
7. Для каждой вершины **j** из множества **J** выполнить процедуру **Visit**.
8. Вершину **k**  окрасить в черный цвет: **C[k] = B**.
9. Увеличить номер шага: **t = t + 1**.
10. Отметить время фиксации вершины: **F[k] = t**.

Если задана стартовая вершина **s**, то алгоритм DFS теперь можно свести к следующим двум шагам.

1. Инициализировать массивы: **С** (все вершины белого цвета), **D** (все расстояния равны бесконечности), **P** (все элементы заполнены символом «пустота»). Установить номер шага: **t = 0**.
2. Выполнить процедуру **Visit**  для вершины **s**.
3. **Задание 4.** Доработайте функцию **DFS**, **для** выполнения топологической сортировки графа. Продемонстрировать работу функции. Копии экрана вставить в отчет.

***Топологическая сортировка*** − это процедура упорядочивания вершин ориентированного графа, не имеющего циклов (ациклического графа). В результате топологической сортировки для вершин графа определяется такой порядок, что если их расположить на рисунке в соответствии с этим порядком сверху вниз, то дуги будут направлены только от верхних вершин к нижним. Обычно после выполнения топологической сортировки вершины переименовываются (перенумеровываются) в соответствии с полученным порядком. После такого переименования граф обладает свойством: начальная вершина каждой дуги имеет номер (имя) меньший, чем номер конечной вершина этой дуги.

Наиболее известны два способа топологической сортировки графа: алгоритмы Демукрона и алгоритм применяющий поиск в глубину.

Используем уже реализованную функцию DFS, добавив в неё вектор topological\_sort, при выходе из вершины будем добавлять её в конец списка. После окончания алгоритма список развернём в противоположном порядке.