Частное учреждение образования

«Колледж бизнеса и права»

|  |  |
| --- | --- |
|  | УТВЕРЖДАЮ  Ведущий методист колледжа  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Е.В. Паскал  « \_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 года |
| Специальность 2-40 01 01 «Программное обеспечение информационных технологий» | Учебная дисциплина «Основы алгоритмизации и программирование» |

**Лабораторная работа № 36**

**Инструкционно-технологическая карта**

Тема: Реализация алгоритмов поиска в ширину в графе, заданном различными способами.

Цель: Научиться создавать алгоритмы поиска в ширину в графе, заданном различными способами.

Время выполнения: 2 часа.

1. **Порядок выполнения работы**
2. Изучить теоретические сведения к лабораторной работе.
3. Разработать на языке С++ программу вывода на экран решения задачи в соответствии с вариантом индивидуального задания, указанным преподавателем.
4. Отлаженную, работающую программу сдать преподавателю. Работу программы показать с помощью самостоятельно разработанных тестов.
5. Ответить на контрольные вопросы.
6. **Теоретические сведения**

**Граф**

**Граф** – совокупность точек, соединенных линиями. Точки называются **вершинами**, или узлами, а линии – **ребрами**, или дугами.

**Степень входа вершины** – количество входящих в нее ребер, а **степень выхода вершины** – количество исходящих из этой вершины ребер.

Граф, содержащий ребра между всеми парами вершин, является **полным графом**.

Есть графы, ребрам которых поставлено в соответствие конкретное числовое значение (вес), они называются **взвешенными графами**, а это значение – **весом ребра**.

Когда у ребра оба конца совпадают, т.е. ребро выходит из вершины и входит в эту же вершину, то такое ребро называется **петлей**.

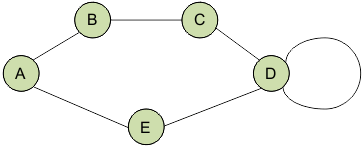


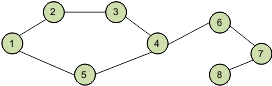
Рисунок 1 – Граф ABCDE с двунаправленными (неориентированными) ребрами и

петлей (ребро справа от вершины D)

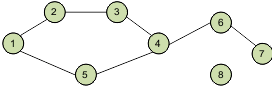
**Классификация графов**

Графы делятся на:

1. Связные



1. Несвязные

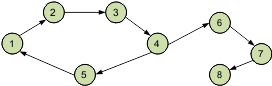


В **связном графе** между любой парой вершин существует как минимум один путь.

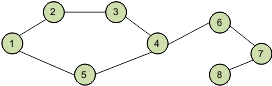
В **несвязном графе** существует хотя бы одна вершина, не связанная с другими вершинами этого графа (в примере вершина № 8).

Графы также подразделяются на:

1. Ориентированные



1. Неориентированные



1. Смешанные.

В **ориентированном графе** ребра являются направленными, т.е. существует только одно доступное направление движения между двумя связными вершинами. В **неориентированном графе** по каждому из ребер можно осуществлять переход в обоих направлениях. Частный случай двух этих видов – **смешанный граф**. Он содержит в себе как ориентированные, так и неориентированные ребра.

**Способы представления графа**

Граф может быть представлен (сохранен) несколькими способами:

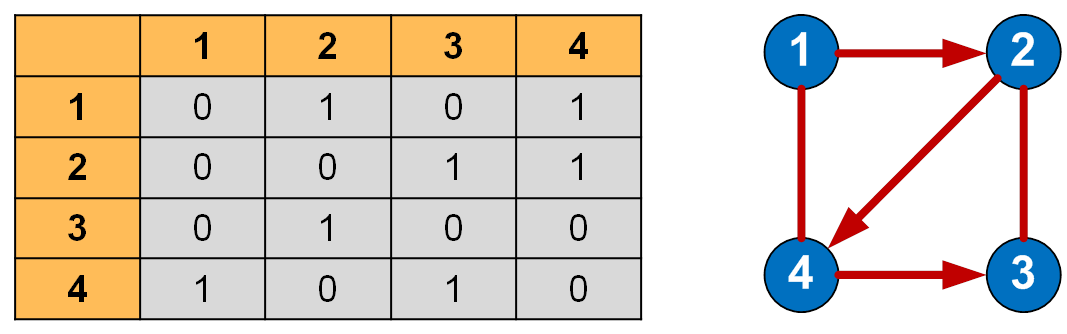
1. матрица смежности;
2. матрица инцидентности;
3. список смежности (инцидентности);
4. список ребер.

Использование двух первых методов предполагает хранение графа в виде двумерного массива (матрицы). Размер массива зависит от количества вершин и (или) ребер в конкретном графе.

**Матрица смежности графа** – это квадратная матрица, в которой каждый элемент принимает одно из двух значений: 0 или 1. Число строк матрицы смежности равно числу столбцов и соответствует количеству вершин графа:

0 – соответствует отсутствию ребра,

1 – соответствует наличию ребра.



Когда из одной вершины в другую проход свободен (имеется ребро), в ячейку заносится 1, иначе – 0. Все элементы на главной диагонали равны 0, если граф не имеет петель.

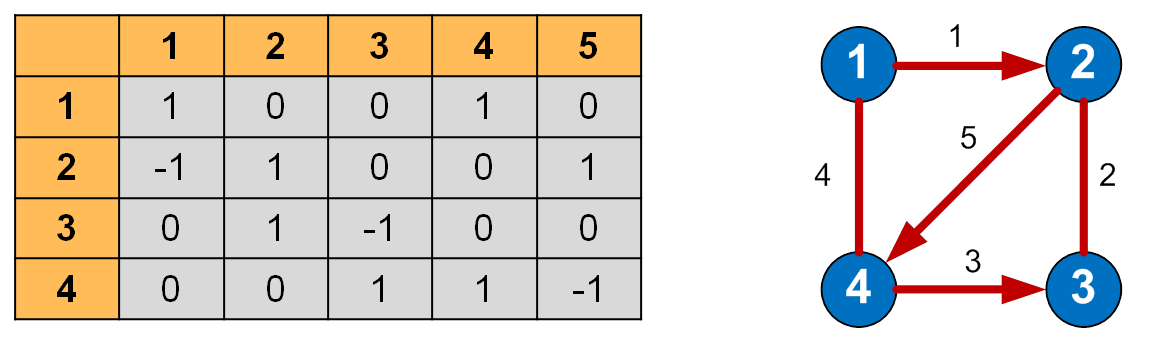
**Матрица инцидентности**

**Матрица инцидентности (инциденции) графа** – это матрица, количество строк в которой соответствует числу вершин, а количество столбцов – числу рёбер. В ней указываются связи между инцидентными элементами графа: ребро (дуга) и вершина.

В неориентированном графе, если вершина инцидентна ребру, то соответствующий элемент равен 1, в противном случае – элемент равен 0.

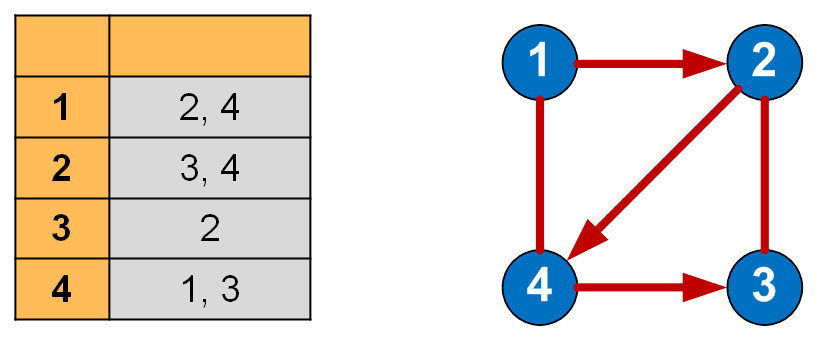
В ориентированном графе, если ребро выходит из вершины, то соответствующий элемент равен 1, если ребро входит в вершину, то соответствующий элемент равен -1, если ребро отсутствует, то элемент равен 0.

Матрица инцидентности для своего представления требует нумерации рёбер, что не всегда удобно.



**Список смежности (инцидентности)**

Если количество ребер графа по сравнению с количеством вершин невелико, то значения большинства элементов матрицы смежности будут равны 0. В таком случае использование данного метода нецелесообразно. Для подобных графов имеются более оптимальные способы их представления. По отношению к памяти компьютера **списки смежности** менее требовательны, чем матрицы смежности. Такой список можно представить в виде таблицы, столбцов в которой – 2, а строк – не больше, чем вершин в графе. В каждой строке в первом столбце указана вершина выхода, а во втором столбце – список вершин, в которые входят ребра из текущей вершины.



**Преимущества списка смежности**:

1. Рациональное использование памяти.
2. Позволяет быстро перебирать соседей у данной вершины.
3. Позволяет проверять наличие ребра и удалять его.

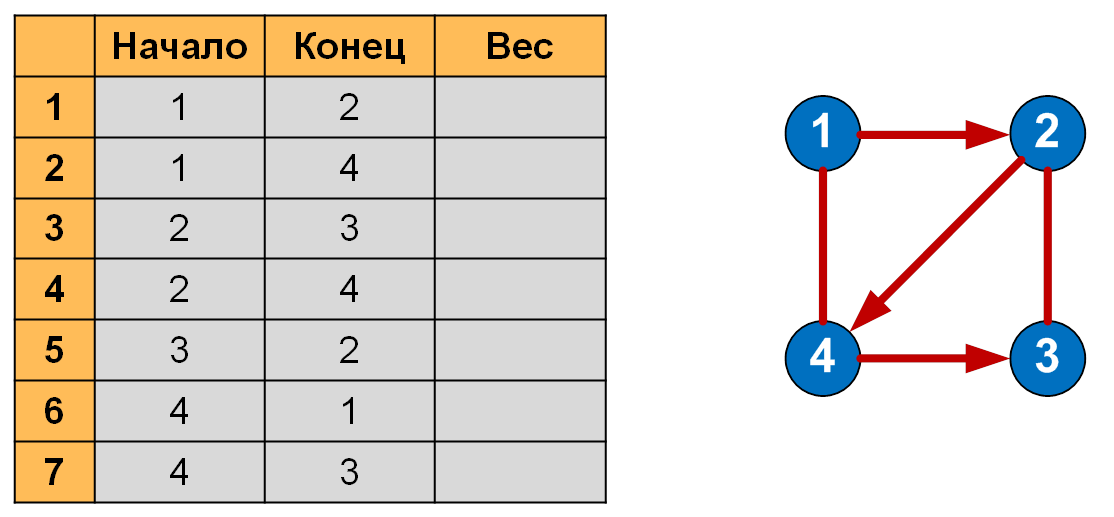
**Недостатки списка смежности**:

1. При работе с **насыщенными графами** (с большим количеством рёбер) скорости может не хватать для обработки всего большого графа.
2. Нет быстрого способа проверить, существует ли ребро между двумя вершинами.
3. Количество вершин графа должно быть известно заранее.
4. Для взвешенных графов приходится хранить список, элементы которого должны содержать два значащих поля (см. далее), что усложняет код:
   1. номер вершины, с которой соединяется текущая вершина;
   2. вес ребра (число).

**Список рёбер**

В **списке рёбер** в каждой строке записываются две смежные вершины и вес соединяющего их ребра (для взвешенного графа).

Количество строк в списке ребер всегда должно быть равно величине, получающейся в результате сложения ориентированных рёбер с удвоенным количеством неориентированных рёбер.



Какой способ представления графа лучше? Ответ зависит от отношения между числом вершин и числом рёбер. Число ребер может быть довольно малым (такого же порядка, как и количество вершин) или довольно большим (если граф является полным). Графы с большим числом рёбер называют **плотными**, с малым – **разреженными**. Плотные графы удобнее хранить в виде матрицы смежности, разреженные – в виде списка смежности.

**Алгоритмы обхода графов**

Основными алгоритмами обхода графов являются:

1. поиск в ширину;
2. поиск в глубину.

**Поиск в ширину** подразумевает поуровневое исследование графа:

1. вначале посещается корень – произвольно выбранный узел,
2. затем – все потомки данного узла,
3. после этого посещаются потомки потомков и т.д.

Вершины просматриваются в порядке возрастания их расстояния от корня.

Алгоритм прекращает свою работу после обхода всех вершин графа, либо в случае выполнения требуемого условия (например, найти кратчайший путь из вершины 1 в вершину 6).

Каждая вершина может находиться в одном из 3-х состояний:

0 – **не**обнаруженная вершина;

1 – обнаруженная, но не посещенная вершина;

2 – обработанная (посещенная) вершина.

**Применения алгоритма поиска в ширину:**

1. Поиск кратчайшего пути в невзвешенном графе (ориентированном или неориентированном).
2. Поиск компонент связности.
3. Нахождения решения какой-либо задачи (игры) с наименьшим числом ходов.
4. Найти все рёбра, лежащие на каком-либо кратчайшем пути между заданной парой вершин.
5. Найти все вершины, лежащие на каком-либо кратчайшем пути между заданной парой вершин.

**Очередь Queue для хранения графов**

**Очередь** – это структура данных, которая построена по принципу **LILO** (Last Input – Last Output: последним пришел – последним обслужишься и последним уйдешь). В C++ уже есть готовый STL контейнер **queue**.

В очереди, если вы добавите элемент, который вошел самый первый, то он и выйдет тоже самым первым. Получается, если вы добавите 4 элемента, то первый добавленный элемент из этих четырех и выйдет тоже самым первым.

Чтобы понять принцип работы очереди вы можете представить себе магазинную очередь. Если вы стоите посреди нее, то чтобы вы оказались напротив кассы, сначала понадобится всех впереди стоящих людей обслужить. А вот для последнего человека в очереди нужно, чтобы кассир обслужил всех людей перед этим человеком, а затем и его самого.

[](https://codelessons.ru/wp-content/uploads/%D0%91%D0%B5%D0%B7%D1%8B%D0%BC%D1%8F%D0%BD%D0%BD%D1%8B%D0%B9-2.jpg)

На рисунке выше находятся 7 чисел: 2, 4, 7, 1, 4, 9, 10. Если нам понадобится их извлечь, то извлекать мы будем эти числа в таком же порядке, как они находятся на рисунке, то есть слева направо. Например, чтобы извлечь число 4 нам понадобится сначала обслужить число 2, а потом уже и само число 4.

Хотя в стеке присутствует функция peek() (она позволяет обратится к элементу по индексу), в шаблоне очереди **не**возможно обратится к **определенному элементу по индексу.**

## **Как создать очередь в С++**

Если вы хотите использовать шаблон очереди в C++, то вам сначала нужно подключить библиотеку queue: #include <queue>.

Дальше для объявления очереди нужно воспользоваться конструкцией queue <тип данных> имяОчереди; , в которой сначала нужно написать слова queue, потом в угловых скобках **< >** мы должны указать тот <типДанных>, элементами которого будем заполнять нашу очередь, и в конце остается только указать название очереди.

Вот пример правильного объявления: queue <int> q;

## **Методы очереди**

**Метод** – это та же самая функция, но она работает только с классами, экземплярами классов, в нашем случае – с контейнерами STL, например, очередью, стеком и т.д.

Для работы с очередью вам понадобится знать методы push() – добавить значение в очередь, pop() – извлечь значение из начала очереди, front() – взять (прочитать) значение из начала очереди, back() – взять (прочитать) значение из конца очереди, empty() – проверка очереди на пустоту (вернет true, если очередь пустая, или вернет false, если очередь не полностью пустая.

Для добавления в очередь нового элемента нужно воспользоваться методом push(). В круглых скобках должно находится значение, которое мы хотим добавить.

Если нам понадобилось удалить первый элемент, то нужно оперировать методом pop(). В круглых скобках ничего не нужно указывать, но по правилам они в обязательном порядке должны присутствовать, как у любого другого метода и функции. Методы empty(), back() и front() тоже не нуждаются в указании входного аргумента.

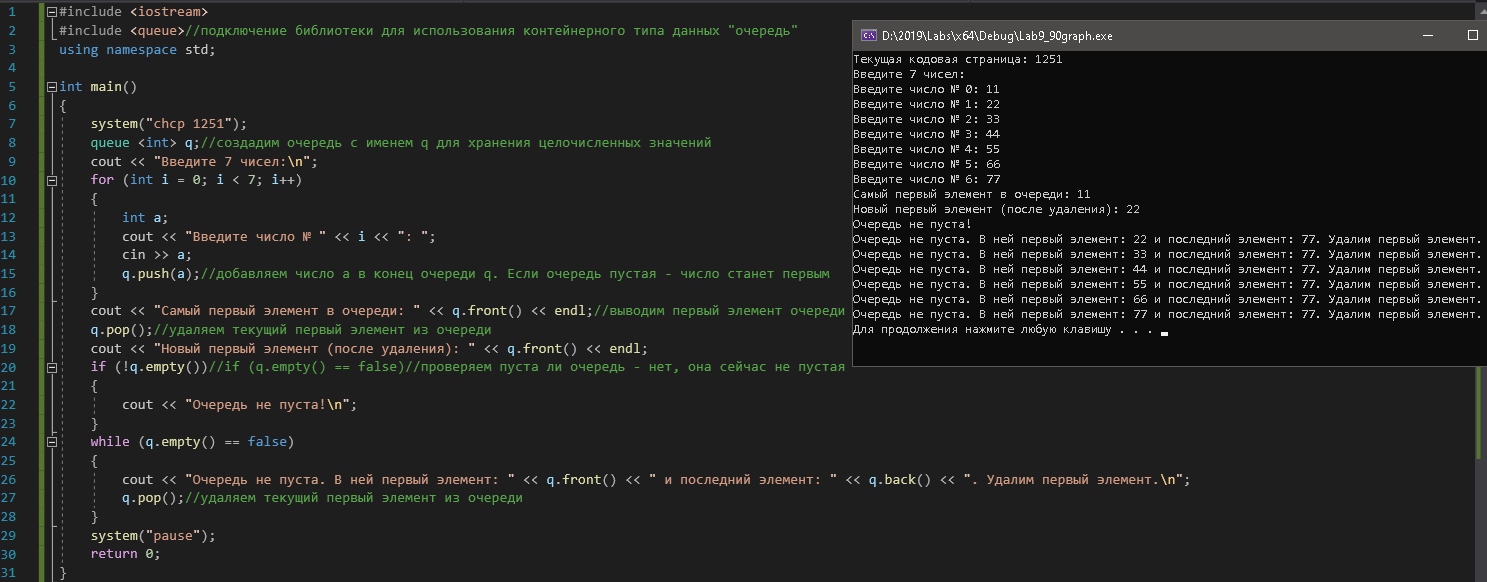
Если вам понадобилось обратиться к первому элементу очереди, то вам понадобится метод front().

Чтобы обратиться к последнему элементу в очереди вам поможет метод back().

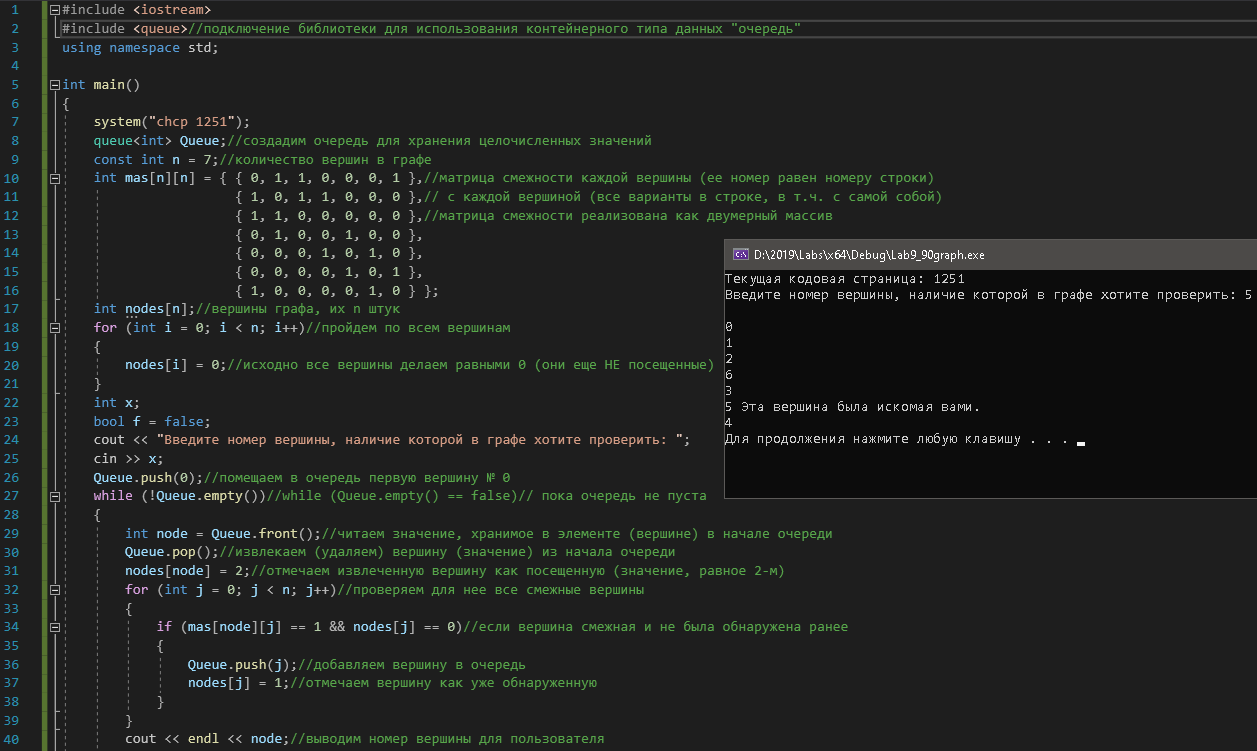
Чтобы узнать, пуста ли очередь, нужно воспользоваться методом empty(), который, если ваша очередь пуста, то возвратит true, а если в ней что-то есть, то возвратит false.

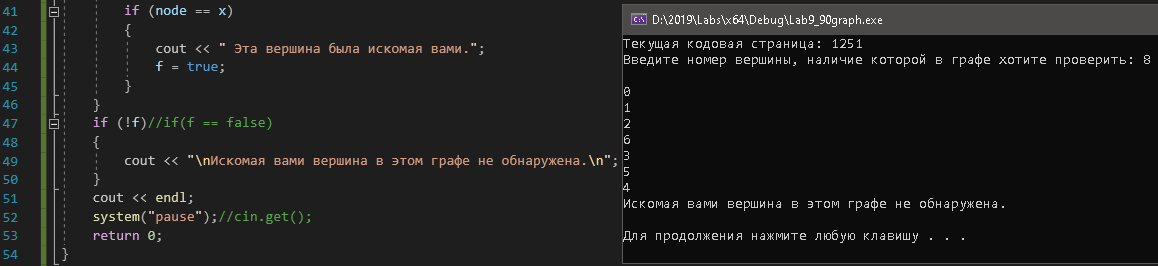
1. **Пример выполнения программы**

Поскольку для работы с графами удобно использовать готовый контейнерный класс очередь (Queue), то потренируемся его создавать, заполнять элементами, читать значения элементов из начала и из конца, удалять элемент из начала очереди, проверять очередь на пустоту:



Алгоритм поиска в ширину работает как на ориентированных, так и на неориентированных графах. Для реализации алгоритма удобно использовать очередь.





1. **Задания по вариантам**

Вывести матрицы смежности, инцидентности, весов для графа, представленного на рисунке по вашему варианту. Реализовать алгоритм поиска в ширину значения элемента, вводимого пользователем с клавиатуры, в вашем графе.

1)

**1**

**2**

**8**

**3**

**42**

**5**

**9**

**7**

**6**

2)

**2**

**3**

**1**

**6**

**5**

**7**

**8**

**4**

3)

**1**

**2**

**3**

**42**

**5**

**6**

**7**

**8**

**9**

4)

**1**

**2**

**8**

**1**

**72**

**39**

**91**

**49**

**42**

**83**

5)

**3**

**2**

**1**

**6**

**5**

**7**

**8**

**4**

6)

**2**

**3**

**1**

**6**

**8**

**5**

**4**

**7**

7)

**7**

**3**

**1**

2

**42**

**5**

**6**

**8**

**9**

8)

**1**

**3**

**5**

**42**

**2**

**6**

**7**

**8**

**9**

9)

**1**

**2**

6

**42**

**5**

**9**

**7**

**3**

8

10)

**2**

**3**

**1**

**6**

**5**

**7**

**8**

**4**

11)

**4**

3

**62**

**5**

**2**

**7**

**8**

**9**

1

12)

**5**

**2**

**8**

**42**

**1**

**6**

**7**

**3**

**9**

13)

**2**

**3**

**1**

**6**

**5**

**7**

**8**

**4**

14)

**2**

**3**

**1**

**6**

**5**

**7**

**8**

**4**

15)

**2**

**3**

**1**

**6**

**5**

**7**

**8**

**4**

1. **Контрольные вопросы**
2. Дайте определение понятию «граф».
3. Дайте определение понятию «ребро графа».
4. Дайте определение понятию «полный граф»?
5. Дайте определение понятию «смешанный граф»?
6. Дайте определение понятию «взвешенный граф»?
7. Что такое «вес ребра»?
8. В чем разница между связным и несвязным графами?
9. В чем разница между ориентированным и неориентированным графами?
10. Что такое матрица смежности и как в нее сохранить данные графа?
11. Что такое матрица инцидентности и как в нее сохранить данные графа?
12. Что такое список смежности (инцидентности) и как в него сохранить данные графа?
13. Что такое список ребер и как в него сохранить данные графа?
14. В чем преимущества и недостатки списка смежности для графа?
15. В чем разница полного и насыщенного графов?
16. В чем разница плотного и разреженного графов?
17. Какие есть способы обхода графов?
18. Опишите поиск в ширину для графа.
19. Что такое очередь queue?
20. Почему для обхода графа в ширину удобно использовать очередь?

**Литература**

**Дейтел,** Х.М. Как программировать на С++ / Х.М. Дейтел, П.Дж. Дейтел . – М. : Бином-Пресс , 2018 . – 1456 с.

**Павловская**, Т.А. С++. Объектно-ориентированное программирование : практикум / Т.А. Павловская, Ю.А. Щупак . – СПб. : Питер , 2019 . – 265 с.

**Страуструп**, Б. Язык программирования С++ / Б. Страуструп . – СПб. : Бином-Пресс , 2019 . – 1054 с.

Преподаватель Шаляпин Ю.В.

|  |
| --- |
| Рассмотрено на заседании цикловой  комиссии ПОИТ № 10  Протокол №\_\_\_\_от «\_\_\_»\_\_\_\_\_\_\_\_\_\_2021 г.  Председатель ЦК \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |