Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К курсовому проектированию

по курсу «Логика и основы алгоритмизации

в инженерных задачах»

на тему «Реализация алгоритма обхода графа в ширину»

Выполнил:

студент группы

Принял:

Пенза 2023

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСТЕТ

\_\_\_\_\_\_\_\_\_ Факультет вычислительной техники\_\_\_\_\_\_\_\_\_\_\_\_

**Кафедра\_\_\_\_\_\_\_\_\_\_ Вычислительная техника**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_

«УТВЕРЖДАЮ»

Зав. Кафедрой\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_» \_декабря\_\_202\_ г.

ЗАДАНИЕ

на курсовое проектирование по курсу

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Студенту\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Группы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Тема проекта\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Исходный данные (технические требования) на проектирование

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Объем работы по курсу

1.Расчетная часть

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2.Графическая часть

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3.Экспрементальная часть.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Срок выполнения проекта по разделам

1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_4\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_5\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания «\_\_\_» \_декабря 202 г.\_

Дата защиты проекта «\_\_\_» \_ 202 г.\_

Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Задание получил «\_\_\_» \_\_\_\_\_Октября 202 г.\_

Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Содержание**

[ВВЕДЕНИЕ](#_Toc422446954)

[1. Основные теоретические данные о графах](#_Toc422446955)

[2. Основные способы «Обхода графа»](#_Toc422446961)

[2.2 «Обход графа в ширину»](#_Toc422446963)

[3. Реализация алгоритма «Обход графа в ширину»](#_Toc422446964)

[3.1 Описание методов и средств реализаци](#_Toc422446965)

[3.2 Програмная реализация «Обход графа в ширину»](#_Toc422446965)

[4 Тестирование алгоритма «Обход графа в ширину»](#_Toc422446967)

З[аключение](#_Toc422446969)

Список используемой литературы

Приложение А

**Введение**

Первая работа по теории графов, принадлежащая известному швейцарскому математику Л.Эйлеру, появилась в 1736г. Вначале теория графов казалась довольно незначительным разделом математики, так как она имела дело в основном с математическими развлечениями и головоломками. Однако дальнейшее развитие математики и особенно её приложений дало сильный толчок развитию теории графов. Уже в XIX столетии графы использовались при построении схем.

В настоящее время эта теория находит многочисленное применение в разнообразных практических вопросах: при установлении разного рода соответствий, при решении транспортных задач, задач о потоках в сети нефтепроводов, в программировании и теории игр, теории передачи сообщений. Теория графов теперь применяется и в таких областях, как экономика, психология и биология.

В этой работе мы подробнее рассмотрим эйлеровы графы, основные сведения и теоремы, связанные с этим понятием. А также задачи, которые решаются с помощью эйлеровых графов.

**1. Основные теоретические данные о графах**

Граф — это множество точек или вершин и множество линий или ребер, соединяющих между собой все или часть этих точек. Вершины, прилегающие к одному и тому же ребру, называются смежными.   
Если ребра ориентированные, что обычно показывают стрелками, то они называются дугами, и граф с такими ребрами называется ориентированным графом. Если ребра не имеют ориентации, граф называется неориентированным (Рисунок 1). Графы обычно изображаются в виде геометрических фигур, так что вершины графа изображаются точками, а ребра - линиями, соединяющими точки.

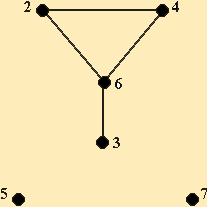


Рисунок 1. Неориентированный граф

Петля это дуга, начальная и конечная вершина которой совпадают.

Степень вершины — это удвоенное количество петель, находящихся у этой вершины плюс количество остальных прилегающих к ней ребер.

Пустым называется граф без ребер. Полным называется граф, в котором каждые две вершины смежные.

Путь в ориентированном графе — это последовательность дуг, в которой конечная вершина всякой дуги, отличной от последней, является начальной вершиной следующей.

Вершины v0, vn называются связанными данным путем (или просто связанными). Вершину v0 называют началом, vn - концом пути. Если v0 = vn, то путь называют **замкнутым**. Число **n** называется длиной пути.

Маршрут в графе —­ путь, ориентацией дуг которого можно пренебречь.   
Цепь — маршрут, в котором все ребра попарно различны.  
Цикл — замкнутый маршрут, являющийся цепью.

Маршрут, в котором все вершины попарно различны, называют **простой цепью**. Цикл, в котором все вершины, кроме первой и последней, попарно различны, называются **простым циклом**. Построим граф (рисунок 2), изображающий отношение делимости на множестве **{1,2,3,4,5,6,7,8,9,10}**. Принцип такой: если от одного числа до другого есть цепь, ведущая вверх, тогда второе число делится на первое.

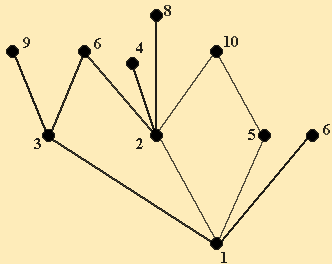
****

Рисунок 2. Маршрут на ориентированном графе

Подграф графа — это граф, являющийся подмоделью исходного графа, т.е. подграф содержит некоторые вершины исходного графа и некоторые ребра (только те, оба конца которых входят в подграф).

Подграф, порожденный множеством вершин **U** это подграф, множество вершин которого - **U**, содержащий те и только те ребра, оба конца которых входят в **U**.

Подграф называется **остовным** подграфом, если множество его вершин совпадает с множеством вершин самого графа.

Граф называется **связным**, если любая пара его вершин связана.   
Связными компонентами графа называются подграфы данного графа, вершины которых связаны.

Дерево — это связный граф без циклов.  
Деревья особенно часто возникают на практике при изображении различных иерархий. Например, популярны генеалогические деревья.

На рисунке 3 показано библейское генеалогическое дерево.

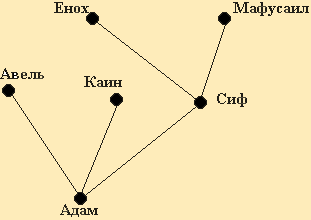


Рисунок 3. Библейское генеалогическое дерево.

Граф без цикла называется **лесом**. Вершины степени 1 в дереве называются **листьями**.   
Деревья — очень удобный инструмент представления информации самого разного вида.   
Деревья отличаются от простых графов тем, что при обходе дереваневозможны циклы. Это делает графы очень удобной формой организации данных для различных алгоритмов. Таким образом, понятия дерева активно используется в информатике и программировании.

Очевидно, что графический способ представления графов непригоден для ПК. Поэтому существуют другие способы представления графов.

В теории графов применяются

1. **Матрица инцинденций**. Это матрица **А** с **n** строками, соответствующими вершинам, и **m** столбцами, соответствующего рёбрам. Для ориентированного графа столбец, соответствующий дуге **(х,y)** содержит - **1** в строке, соответствующей вершине **х** и **1**, в строке, соответствующей вершине **у**. Во всех остальных **0**. Петлю, т.е. дугу **(х,х)** можно представлять иным значением в строке **х**, например, **2**. Если граф неориентированный, то столбец, соответствующий ребру **(х,у)** содержит **1**, соответствующие **х** и **у** и нули во всех остальных строках.
2. **Матрица смежности**. Это матрица **n×n** где **n** - число вершин, где **b**ij **= 1**, если существует ребро, идущее из вершины х в вершину у и **b**ij = 0 в противном случае.

Составим матрицы инциндентности (рисунок 5) и смежности (рисунок 6) для следующего непрерывного графа (рисунок 4).

|  |  |  |
| --- | --- | --- |
| http://matmetod-popova.narod.ru/theme213/picture2_13_4.gif  Рисунок 4. Непрерывный граф | http://matmetod-popova.narod.ru/theme213/matrix_incidence.GIF  Рисунок 5. Матрица инцидентности | http://matmetod-popova.narod.ru/theme213/matrix_adjacency.GIF  Рисунок 6. Матрица смежности |

**Реализация алгоритма «Обход графа в ширину»**

Теперь перейдем к более формальному описанию алгоритма поиска в ширину. Основными объектами – тремя структурами данных, задействованными в программе, будут:

* матрица смежности графа GM;
* очередь queue;
* массив посещенных вершин visited.

Две первые структуры имеют целочисленный тип данных, последняя – логический. Посещенные вершины, заносятся в массив visited, что предотвратит зацикливание, а очередь queue будет хранить задействованные узлы. Напомним, что структура данных «очередь» работает по принципу «первый пришел – первый вышел». Рассмотрим разбитый на этапы процесс обхода графа:

1. массив visited обнуляется, т. е. ни одна вершина графа еще не была посещена;
2. в качестве стартовой, выбирается некоторая вершина s и помещается в очередь (в массив queue);
3. вершина s исследуется (помечается как посещенная), и все смежные с ней вершины помещаются в конец очереди, а сама она удаляется;
4. если на данном этапе очередь оказывается пустой, то алгоритм прекращает свою работу; иначе посещается вершина, стоящая в начале очереди, помечается как посещенная, и все ее потомки заносятся в конец очереди;
5. пункт 4 выполняется пока это возможно.

Поиск в ширину, начиная со стартовой вершины, постепенно уходит от нее все дальше и дальше, проходя уровень за уровнем. Получается, что по окончанию работы алгоритма будут найдены все кратчайшие пути из начальной вершины до каждого доступного из нее узла.

Для реализации алгоритма на языке программирования потребуется: умение программно задавать граф, а также иметь представление о такой структуре данных, как очередь.

Граф, нач. вершина

Q<=пусто

Q<=нач. вершина

W!= visited

Q<=W

(V,W)= visited(дуга)

W!=0

W – не окраш. Вершина, смежная вершина V W=0(такой верш. нет)

Окрасить нач. вершину

Q!=пусто

нет

нет

да

нет

Рисунок 4.1 - Блок – схема алгоритма «Обход графа в ширину»

да

окраш. верш. и дуги

W=visited(старт)

1. **Описание методов и средств реализации**

Программа, выполняющая действия над матрицами, реализована с помощью динамических массивов (так как есть необходимость задавать матрицы произвольной размерности) и функций. Использование функций в программе наглядно показано на блок – схеме.

Динамический массив в С++

Динамическое выделение памяти необходимо для эффективного использования памяти компьютера. Например, мы написали какую-то программку, которая обрабатывает массив. При написании данной программы необходимо было объявить массив, то есть задать ему фиксированный размер (к примеру, от 0 до 100 элементов). Тогда данная программа будет не универсальной, ведь может обрабатывать массив размером не более 100 элементов. А если нам понадобятся всего 20 элементов, но в памяти выделится место под 100 элементов, ведь объявление массива было статическим, а такое использование памяти крайне неэффективно.

В С++ операции new и delete предназначены для динамического распределения памяти компьютера.  Операция new выделяет память из области свободной памяти, а операция n delete высвобождает выделенную память.  Выделяемая память, после её использования должна высвобождаться, поэтому операции new и delete используются парами. Даже если не высвобождать память явно, то она освободится ресурсами ОС по завершению работы программы. Рекомендую все-таки не забывать про операцию delete.

Операция new создает объект заданного типа, выделяет ему память и возвращает указатель правильного типа на данный участок памяти. Если память невозможно выделить, например, в случае отсутствия свободных участков, то возвращается нулевой указатель, то есть указатель вернет значение 0. Выделение памяти возможно под любой тип данных: int, float, double, char и т.д.

Как было сказано раньше, массивы также могут быть динамическими. Чаще всего операции new и delete применяются, для создания динамических массивов, а не для создания динамических переменных.

После того как динамический массив стал ненужным, нужно освободить участок памяти, который под него выделялся.

Помимо одномерных динамических массивов, также можно создавать и двумерные (матрицы). Объявление и удаление двумерного динамического массива выполняется с помощью цикла.

**Функции**

Простые программы мы писали единым, функционально неделимым, кодом. Алгоритм программы находился в главной функции, причём других функций в программе не было. Мы писали маленькие программы, поэтому не было потребности в объявлении своих функций. Для написания больших программ, опыт показывает, что лучше пользоваться функциями. Программа будет состоять из отдельных фрагментов кода, под отдельным фрагментом кода понимается функция. Отдельным, потому, что работа отдельной функции не зависит от работы какой-нибудь другой. То есть алгоритм в каждой функции функционально достаточен и не зависим от других алгоритмов программы. Однажды написав функцию, её можно будет с лёгкостью переносить в другие программы. Функция (в программировании) — это фрагмент кода или алгоритм, реализованный на каком-то языке программирования, с целью выполнения определённой последовательности операций. Итак, функции позволяют сделать программу модульной, то есть разделить программу на несколько маленьких подпрограмм (функций), которые в совокупности выполняют поставленную задачу. Еще один огромнейший плюс функций в том, что их можно многократно использовать. Данная возможность позволяет многократно использовать один раз написанный код, что в свою очередь, намного сокращает объем кода программы!

Кроме того, что в С++ предусмотрено объявление своих функций, также можно воспользоваться функциями, определёнными в стандартных заголовочных файлах языка программирования С++. Чтобы воспользоваться функцией, определённой в заголовочном файле, нужно его подключить. Например, чтобы воспользоваться функцией, которая возводит некоторое число в степень, нужно подключить заголовочный файл <cmath> и в запустить функцию pow() в теле программы. Разработаем программу, в которой запустим функцию pow().

Функций своего типа бывает два:

* Функции, которые не возвращают значение;
* Функции, возвращающие значение.

Функции, не возвращающие значения, завершив свою работу, никакого ответа программе не дают. Рассмотрим структуру объявления таких функций.

Пример объявления структуры функции, не возвращающей значение:

1. void /\*имя функции\*/(/\*параметры функции\*/) // заголовок функции

2. {

3. //тело функции

4. }

Строка 1 начинается с зарезервированного слова void — это тип данных, который не может хранить какие-либо данные. Тип данных void говорит о том, что данная функция не возвращает никаких значений. void никак по-другому не используется и нужен только для того, чтобы компилятор мог определить тип функции. После зарезервированного слова void пишется имя функции. Сразу за именем функции ставятся две круглые скобочки, открывающаяся и закрывающаяся. Если нужно функции передавать какие-то данные, то внутри круглых скобочек объявляются параметры функции, они отделяются друг от друга запятыми. Строка 1 называется заголовком функции. После заголовка функции пишутся две фигурные скобочки, внутри которых находится алгоритм, называемый телом функции.

Функции, возвращающие значение, по завершению своей работы возвращают определённый результат. Такие функции могут возвращать значение любого типа данных. Структура функций, возвращающих значение будет немного отличатся от структуры функций, рассмотренных ранее.

Пример объявления структуры функции, возвращающей значение:

1. /\*возвращаемый тип данных\*/ /\*имя функции\*/(/\*параметры функции\*/) // заголовок функции

2. {

3. // тело функции

4. return /\*возвращаемое значение\*/;

5. }

**3.2 Программная реализация «Обход графа в ширину»**

Листинг 1: создание очереди обхода

void BFS(bool \*visited, int unit)

{

int \*queue=new int[n];

int count, head;

for (i=0; i<n; i++) queue[i]=0;

count=0; head=0;

queue[count++]=unit;

visited[unit]=true;

На данном участке программы мы задаем массив и делаем на него указатель, а так же проходим по всем элементам массива и заносим их в массив queue, который представляет собой очередь обхода.

Листинг 2: Проверка на посещенность текущей вершины графа

while (head<count)

{

unit=queue[head++];

cout<<unit+1<<" ";

for (i=0; i<n; i++)

if (GM[unit][i] && !visited[i])

{

queue[count++]=i;

visited[i]=true;

}

}

delete []queue;

}

На данном участке кода мы проверяем была ли посещена вершина, в случае если нет мы заносим ее в массив visited и увеличиваем текущий элемент на 1

Листинг 3: Ввод матрицы смежности

setlocale(LC\_ALL, "Rus");

int start;

cout<<"Введите матрицу смежности";

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

cin>>GM[i][j];

}

}

Тут реализован ввод матрицы смежности

Листинг 4: Обход графа в ширину

cout<<"Стартовая вершина >> "; cin>>start;

bool \*visited=new bool[n];

cout<<"Матрица смежности графа: "<<endl;

for (i=0; i<n; i++)

{

visited[i]=false;

for (j=0; j<n; j++)

cout<<" "<<GM[i][j];

cout<<endl;

}

cout<<"Порядок обхода: ";

BFS(visited, start-1);

delete []visited;

system("pause>>void");

}

Реализация обхода матрицы в ширину с последующим выводом на экран порядка обхода графа.

Полную версию кода можно увидеть в Приложении А.

1. **Тестирование программы «Обход графа в ширину»**

На рисунке 17 ниже представлен скриншот рабочей программы:

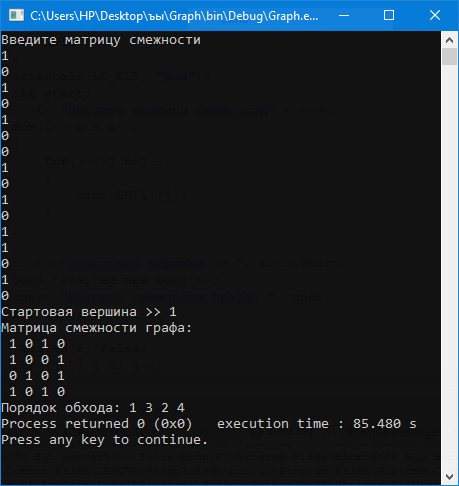


Рисунок 17. Скриншот программы

На первом этапе, пользователю предложено ввести матрицу смежности, согласно которому будет построена связь между вершинами графа. После этого, требуется задать стартовую вершину, согласно которой будет произведен обход графа в ширину и последующий вывод порядка обхода графа.

**Заключение**

В ходе работы были изучены теоретические основы по теме “Графы” , его виды и основные операции над ними .После этого, мы перешли к практической части работы . Был разработан и изучен алгоритм обхода графа в ширину и создана блок – схема в общем и углубленном виде.

Данный алгоритм был реализован на языке программирования C++ ,а затем проверен на наличие ошибок. Мы осуществили ручной просчет ,чтобы убедится в правильности работы нашей программы. Основными структурами, при создании программы являлись : динамические массивы и циклы(while , for) ,а так же функции. В ходе учебной практики, была создана презентация на полученную нами тему . Итогами практических занятий и работой на них - удовлетворены

**Список использованной литературы :**

1. Оре О. «Графы и их применения», М. «Мир», 1965;

2.Зыков А. А. «Теория конечных графов», Новосибирск, «Наука»,1969;

3. Берж К. «Теория графов и ее применение», М., ИЛ, 1962;

4. Реньи А., «Трилогия о математике», М., «Мир», 1980.

5. HYPERLINK «http://www.xumuk.ru» http://www.xumuk.ru

6. HYPERLINK «http://www.seznaika.ru» http://www.seznaika.ru

7. HYPERLINK «http://lib.repetitors.eu» http://lib.repetitors.eu

Приложение A :

Программный код реализации алгоритма обхода графа в ширину

#include <iostream>

#include <iostream>

using namespace std;

const int n=4;

int i, j;

int GM[n][n];

//поиск в ширину

void BFS(bool \*visited, int unit)

{

int \*queue=new int[n];

int count, head;

for (i=0; i<n; i++) queue[i]=0;//проходит по всем строкам, находит текущий элемент, аносит его в очередь, а потом в масси visited

count=0; head=0;

queue[count++]=unit;

visited[unit]=true;

while (head<count)

{

unit=queue[head++];

cout<<unit+1<<" ";

for (i=0; i<n; i++)

if (GM[unit][i] && !visited[i])

{

queue[count++]=i;

visited[i]=true;

}

}

delete []queue;

}

//главная функция

int main()

{

setlocale(LC\_ALL, "Rus");

int start;

cout<<"Введите матрицу смежности" <<endl;

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

cin>>GM[i][j];

}

}

cout<<"Стартовая вершина >> "; cin>>start;

bool \*visited=new bool[n];

cout<<"Матрица смежности графа: "<<endl;

for (i=0; i<n; i++)

{

visited[i]=false;

for (j=0; j<n; j++)

cout<<" "<<GM[i][j];

cout<<endl;

}

cout<<"Порядок обхода: ";

BFS(visited, start-1);

delete []visited;

system("pause>>void");

}