|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 2.5** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Основные алгоритмы работы с графами»** | |
|  | |
| Выполнил студент группы ИКБО-30-22 | Сенькевич Г.Д. |
| Принял преподаватель | Красников С.А. |

Москва 2023

# **Цель работы**

Получить знания и навыки реализации графов и алгоритмов работы с ними.

1. **Ход работы**

# **. Формулировка задачи**

Составить программу создания графа и реализовать процедуру для работы с графом, определенную индивидуальным вариантом задания.

Самостоятельно выбрать и реализовать способ представления графа в памяти.

В программе предусмотреть ввод с клавиатуры произвольного графа. В вариантах построения остовного дерева также разработать доступный способ (форму) вывода результирующего дерева на экран монитора.

Провести тестовый прогон программы на предложенном в индивидуальном варианте задания графе. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.

Сделать выводы о проделанной работе, основанные на полученных результатах.

Оформить отчет с подробным описанием рассматриваемого графа, принципов программной реализации алгоритмов работы с графом, описанием текста исходного кода и проведенного тестирования программы.

Вариант №20. Условие задания:

|  |  |  |
| --- | --- | --- |
| Вариант | Алгоритм | Предложенный граф |
| 20 | Найти и вывести эйлеров цикл в графе. Реализовать обход |  |

* 1. **. Подход к решению**

Граф является абстрактной математической структурой, используемой для представления отношений между объектами. Он состоит из вершин (или узлов) и рёбер (или связей), которые соединяют эти вершины. Графы могут быть направленными или ненаправленными, в зависимости от того, имеют ли рёбра определенное направление.

Существуют различные способы представления графов в памяти компьютера. Один из них - матрица смежности, где строки и столбцы матрицы соответствуют вершинам графа, а элементы матрицы указывают на наличие или отсутствие ребра между соответствующими вершинами.

Работа с графами включает в себя множество операций, таких как добавление и удаление вершин или рёбер, поиск кратчайших путей, проверка связности графа и многое другое. Для выполнения этих операций используются различные алгоритмы.

Обход графа в ширину (BFS - Breadth-First Search) является одним из базовых алгоритмов для обработки графов. Он позволяет обойти все вершины графа, начиная с заданной вершины, поэтапно расширяя область поиска на все ближайшие вершины перед переходом к следующему уровню. Этот метод используется, например, для поиска кратчайшего пути в невзвешенном графе.

Алгоритм поиска эйлерова цикла в графе направлен на нахождение цикла, который проходит через каждое ребро графа ровно один раз. Это может быть полезно для нахождения оптимального маршрута, проходящего через все рёбра графа.

* 1. **. Код программы**

Для решения поставленных задач был написан класс graph (см. листинг 1), реализующий хранение неориентированного графа и работу с ним.

Класс graph содержит следующие (открытые) методы:

* graph – создание графа с заданным количеством вершин;
* insert\_edge – вставка ребра в граф;
* print\_adjacency\_matrix – вывод матрицы смежности;
* print\_bfs – вывод обхода в ширину с началом в указанной вершине;
* is\_connected – проверка графа на связность;
* vertex\_degree – вычисление степени указанной вершины;
* has\_euler\_cycle – проверка графа на наличие ейлерового цикла;
* print\_euler – вывод ейлерового цикла (при наличии);

Описание алгоритма работы print\_bfs:

Обход графа в ширину, реализованный в этой функции, осуществляется следующим образом: создаются массив посещённых вершин и очередь вершин на обработку, в очередь помещается начальная вершина, после чего, пока очередь не пуста, из неё извлекается элемент, все его не посещённые соседи добавляются в очередь и отмечаются посещёнными, а сам он выводится на экран.

Описание алгоритма работы print\_euler:

Эйлеров цикл в графе ищется рекурсивно: функция начинает со случайно выбранной вершины, после чего, пока у неё есть смежные вершины, выполняет следующее действие – удаляет ребро между начальной вершиной и смежной, после чего рекурсивно вызывает функцию от смежной вершины, и когда все смежные вершины закончились, на экран выводится начальная вершина.

Листинг 1 ­— Класс graph.

|  |
| --- |
| // Класс, описывающий неориентированный граф  class graph  {  private:  // Количество рёбер  int edges = 0;  // Количество вершин  int vertices = 0;  // Матрица смежности  std::vector<std::vector<bool>> adjacency\_matrix;  // Всомогательная функция для print\_euler  // (выполняет основную работу)  void print\_euler\_util(int start, std::vector<std::vector<bool>>& matrix)  {  // Цикл по всем вершинам  for (int i = 0; i < vertices; i++)  {  // Если нашли смежную вершину с начальной  if (matrix[start][i])  {  // Удаляем ребро между этой вершиной и начальной  matrix[start][i] = false;  matrix[i][start] = false;  // Рекурсивно запускаем функцию от этой вершины  print\_euler\_util(i, matrix);  }  }  // Выводим начальную вершину  std::cout << start << ' ';  }  public:  // Конструктор (создаёт матрицу смежности и заполняет её false)  graph(int vertices) : adjacency\_matrix(vertices)  {  this->vertices = vertices;  for (std::vector<bool>& row : adjacency\_matrix)  {  row = std::vector<bool>(vertices, false);  }  }  // Вставляет ребро в граф  void insert\_edge(int first, int second)  {  if (first < 0 || first >= vertices ||  second < 0 || second >= vertices)  {  return;  }  // Вставляем ребро first->second  adjacency\_matrix[first][second] = true;  // Вставляем ребро second->first  adjacency\_matrix[second][first] = true;  edges++;  }  // Выводит матрицу смежности  // 0 - false, 1 - true  void print\_adjacency\_matrix()  {  for (int i = 0; i < vertices; i++)  {  for (int j = 0; j < vertices; j++)  {  std::cout << (adjacency\_matrix[i][j] ? 1 : 0) << ' ';  }  std::cout << '\n';  }  }  // Выводит обход графа в ширину  void print\_bfs(int start)  {  if (start < 0 || start >= vertices)  {  return;  }  // Список посещённых вершин  std::vector<bool> visited(vertices, false);  // Очередь вершин на обработку  std::queue<int> queue;  // Помещаем в очередь начальную вершину  queue.push(start);  visited[start] = true;  // Пока в очереди есть вершины  while (!queue.empty())  {  // Берём первую вершину из очереди  int current = queue.front();  std::cout << current << ' ';  // Цикл по всем вершинам  for (int i = 0; i < vertices; i++)  {  // Если вершина смежна с текущей и не посещена  if (adjacency\_matrix[current][i] && !visited[i])  {  // Отмечаем её посещённой и добавляем в очередь  visited[i] = true;  queue.push(i);  }  }  // Удаляем текущую вершину из очереди  queue.pop();  }  std::cout << '\n';  }  // Проверяет граф на связность  bool is\_connected()  {  if (vertices == 0)  {  return true;  }  // Список посещённых вершин  std::vector<bool> visited(vertices, false);  // Очередь вершин на обработку  std::queue<int> queue;  // Помещаем в очередь начальную вершину  queue.push(0);  visited[0] = true;  // Пока в очереди есть вершины  while (!queue.empty())  {  // Берём первую вершину из очереди  int current = queue.front();  // Цикл по всем вершинам  for (int i = 0; i < vertices; i++)  {  // Если вершина смежна с текущей и не посещена  if (adjacency\_matrix[current][i] && !visited[i])  {  // Отмечаем её посещённой и добавляем в очередь  visited[i] = true;  queue.push(i);  }  }  // Удаляем текущую вершину из очереди  queue.pop();  }  // Цикл по всем вершинам  for (int i = 0; i < visited.size(); i++)  {  // Если вершина не была посещена в ходе обхода  if (!visited[i])  {  // Граф несвязный  return false;  }  }  // Если все вершины были посещены,  // граф связный  return true;  }  // Вычисляет степень вершины  int vertex\_degree(int vertex)  {  if (vertex < 0 || vertex >= vertices)  {  return -1;  }  int degree = 0;  // Цикл по всем вершинам  for (int i = 0; i < vertices; i++)  {  // Если вершина смежна с рассматриваемой  if (adjacency\_matrix[vertex][i])  {  // Увеличиваем степень на 1  degree++;  }  }  // Возвращаем степень  return degree;  }  // Прооверяет наличие ейлерова цикла  bool has\_euler\_cycle()  {  // Ейлеров цикл может существовать  // только в связных графах  if (!is\_connected())  {  return false;  }  // Каждая вершина должна иметь  // чётную степень  for (int i = 0; i < vertices; i++)  {  if (vertex\_degree(i) % 2 == 1)  {  return false;  }  }  return true;  }  // Печатает ейлеров цикл  void print\_euler()  {  // Если в графе есть ейлеров цикл  if (has\_euler\_cycle())  {  // Записываем матрицу смежности в отдельную переменную  // (она будет изменяться)  std::vector<std::vector<bool>> matrix = adjacency\_matrix;  // Вызываем вспомогательную функцию  print\_euler\_util(0, matrix);  std::cout << '\n';  }  else  {  std::cout << "В этом графе нет ейлерова цикла!\n";  }  }  }; |

* 1. **. Результаты тестирования**

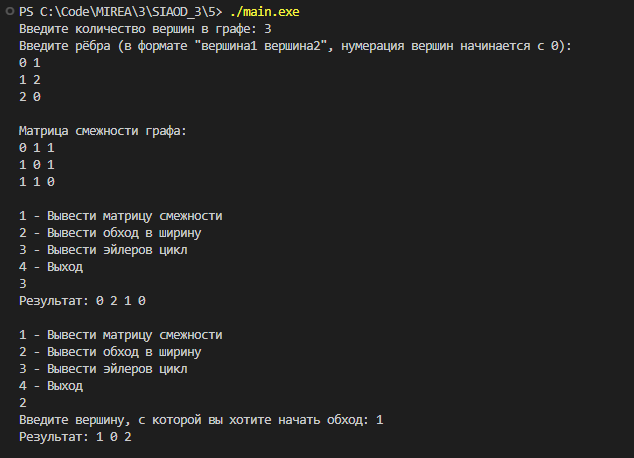


Рисунок 1 – Тестирование всех функций на простом примере (K3)

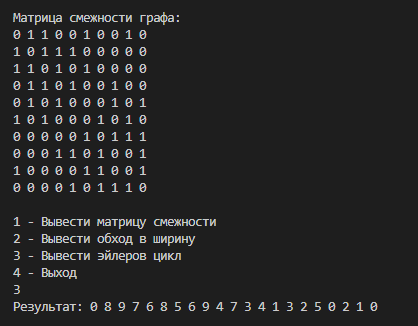
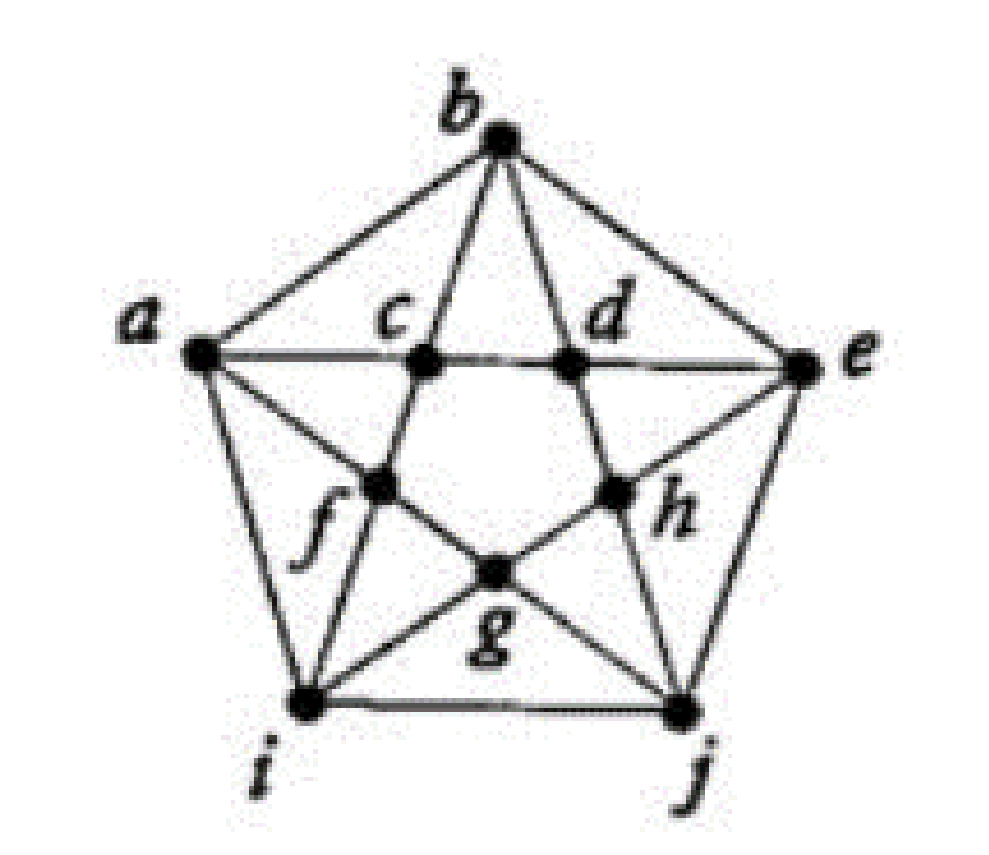


Рисунок 2 –Тестирование на графе из примера.

На рисунке 1 представлено тестирование всех функций программы на полном графе из 3 вершин. Как видно, все они дают корректный результат.

На рисунке 2 выводится эйлеров цикл графа из примера. Обозначения a-j заменены на 0-9, соответственно в исходных обозначениях цикл выглядел бы так:



**a i j h g i f g j e h d e b d c f a c b a**

Тестирование показало, что программа работает правильно, корректно решая поставленные задачи.

1. **Выводы**

В результате выполнения работы я:

1. Изучил основные алгоритмы работы с графами;
2. Реализовал граф и некоторые алгоритмы работы с ним на C++.
3. **Список литературы**
4. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.
5. Документация по языку С++ [Электронный ресурс]. URL: https://docs.microsoft.com/ruru/cpp/cpp/ (дата обращения 01.09.2021).
6. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 01.09.2021).