Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

**«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПЕРМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра “Информационные технологии и автоматизированные системы”

**ОТЧЕТ**

**по лабораторной работе №21**

Дисциплина: “Основы теории алгоритмов и структуры данных”

Тема: Графы (Алгоритм Дейкстры)

Вариант №11

Выполнил:

Студент группы ИВТ-20-2б

Сабуров Павел Алексеевич

Проверила:

Доцент кафедры ИТАС

Полякова О.А.

**Пермь, 2021**

**Цель работы**

Продемонстрировать работу алгоритма Дейкстры на заданном графе

**Постановка задачи**

Реализовать алгоритм Дейкстры для графа, соответствующего выбранному варианту.

1. Интерфейс на усмотрение разработчика.

2 Визуализация графа в Qt.

3. Реализуется только метод Дейкстры

**Анализ задачи**

**(1)**

1. Ввод стартовой вершины, добавление в очередь обрабатываемых вершин

В цикле (рассмотрена одна из итераций):

1. Достать вершину из очереди для обработки
2. Заполнить вектор соседей
3. Расположение соседей в порядке возрастания
4. Переопределение веса вершины каждого соседа вершины “x” и добавление в очередь на обработку
5. Обозначение конца обработки вершины “x” и удаление из очереди
6. Вывод кратчайших путей от вершины “x” до остальных вершин

**(2)**

int x; - вершина в главной роли

int len - количество вершин в графе

int result[6] - массив расстояний от вершины “x” до остальных вершин

bool flag[6] - массив обозначений обработанных вершин

vector <vector<int>> matrix - матрица смежности

queue <int> QUEUE; - очередь вершин на обработку

vector <int> vec; - вектор соседей у вершины “x”

**(3)**

Элементы графа представлены в виде матрицы смежности:

|  |
| --- |
| vector <vector<int>> matrix = { {0 ,14 ,0,0,0 ,0 },  {0,0 ,0 ,0,42 ,23 },  {19 ,0 ,0 ,9 ,0,0 },  {0 ,0 ,0 ,0 ,0,31},  {0 ,0,18 ,0,0 ,0 },  {28,23 ,0 ,0 ,0 ,0 } }; |

**(4)**

Ввод в матрицу происходит руками данными из варианта 21 лабораторной работы

Вывод кратчайшего пути из заданной вершины в остальные происходит следующим образом:

|  |
| --- |
| cout << "Кратчайшие пути:\n";  for (int i = 0; i < len; i++)  {  if (result[i] == 10000) cout << i + 1 << " - недоступен\n";  else cout << i + 1 << " - " << result[i] << endl;  } |

**(5)** Поставленные задачи стоит решать следующим образом:

1. Ввод стартовой вершины, добавление в очередь обрабатываемых вершин:

|  |
| --- |
| cin >> x;  x--;  result[x] = 0;  QUEUE.push(x); |

Вершина вводится, далее декрементируется (т.к. индексы на единицу меньше порядковых номеров). Далее данной вершине присваивается в результирующий массив 0, т.к. пользователь уже находится в ней. Следующим действием вершина добавляется в очередь на обработку

1. Достать вершину из очереди для обработки:

|  |
| --- |
| x = QUEUE.front(); |

1. Заполнить вектор соседей вершины “x”:

|  |
| --- |
| vector <int> vec;  for (int i = 0; i < len; i++) if (matrix[x][i] != 0 && !flag[i]) vec.push\_back(i); |

Объявляется вектор соседей. Далее по строке матрицы смежности у вершины “x” заполняется вектор соседей, параллельно проверяя соседа на “еще не посещенную” и не соответствующую самой вершине “x”

1. Расположение соседей в порядке возрастания:

|  |
| --- |
| sort(vec.begin(), vec.end(), comp); |

Используется функция из библиотеки <algorithm>, и сортировка осуществляется посредством сравнения двух соседних элементов вектора:

|  |
| --- |
| bool comp(int a, int b) {  return matrix[x][a] < matrix[x][b]; } |

1. Переопределение веса вершины каждого соседа вершины “x” и добавление в очередь на обработку:

|  |
| --- |
| for (int i = 0; i < vec.size(); i++)  {  result[vec[i]] = min(result[vec[i]], matrix[x][vec[i]] + result[x]);  QUEUE.push(vec[i]);  } |

У каждой вершины из вектора соседей переопределяется вес на сумму веса вершины “x” и пути из нее в текущего соседа вершины “x” или остается прежним. Далее сосед добавляется в очередь на обработку

1. Обозначение конца обработки вершины “x” и удаление из очереди:

|  |
| --- |
| flag[x] = true; QUEUE.pop(); |

Обработанная вершина помечается посещенной. Далее она удаляется из очереди, будучи обработанной

1. Вывод кратчайших путей от вершины “x” до остальных вершин:

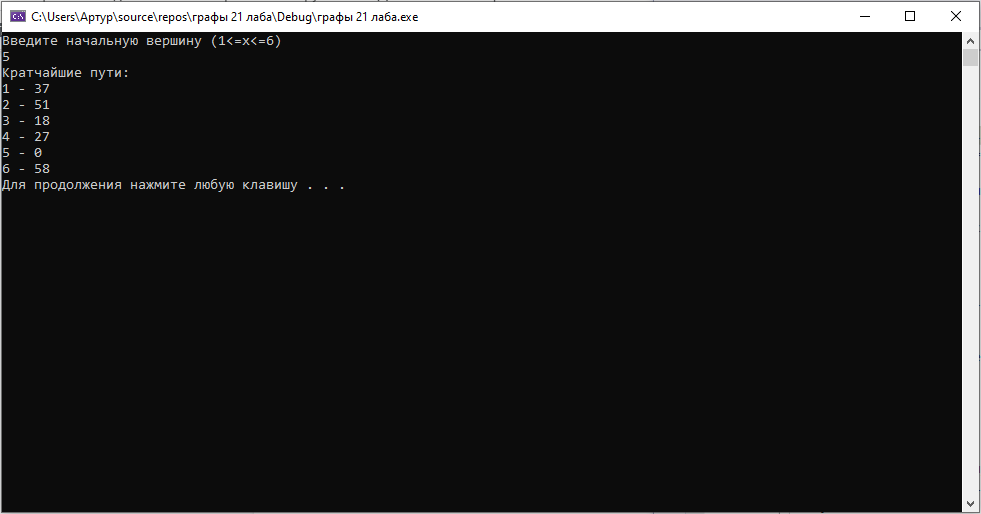
|  |
| --- |
| cout << "Кратчайшие пути:\n";  for (int i = 0; i < len; i++)  {  if (result[i] == 10000) cout << i + 1 << " - недоступен\n";  else cout << i + 1 << " - " << result[i] << endl;  } |

Проходя по результирующему массиву выводятся расстояния от заданной вершины до остальных вершин

**Код на C++**

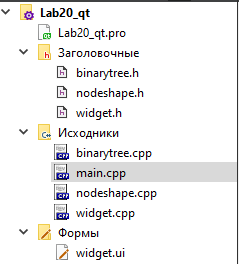
|  |
| --- |
| #include <iostream> #include <vector> #include <windows.h> #include <queue> #include <algorithm> using namespace std; int x; int len = 6; int result[6] = { 10000,10000,10000,10000,10000,10000 }; bool flag[6] = { false,false,false,false,false,false }; vector <vector<int>> matrix = { {0 ,14 ,0,0,0 ,0 },  {0,0 ,0 ,0,42 ,23 },  {19 ,0 ,0 ,9 ,0,0 },  {0 ,0 ,0 ,0 ,0,31},  {0 ,0,18 ,0,0 ,0 },  {28,23 ,0 ,0 ,0 ,0 } }; bool comp(int a, int b) {  return matrix[x][a] < matrix[x][b]; } int main() {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  queue <int> QUEUE;  cout << "Введите начальную вершину (1<=x<=" << len << ")\n";  cin >> x;  x--;  result[x] = 0;  QUEUE.push(x);  while (true)  {  x = QUEUE.front();  vector <int> vec;  for (int i = 0; i < len; i++) if (matrix[x][i] != 0 && !flag[i]) vec.push\_back(i);  if (vec.size() == 0)  {  flag[x] = true;  QUEUE.pop();  if (QUEUE.size() == 0) break;  continue;  }  sort(vec.begin(), vec.end(), comp);  for (int i = 0; i < vec.size(); i++)  {  result[vec[i]] = min(result[vec[i]], matrix[x][vec[i]] + result[x]);  QUEUE.push(vec[i]);  }  flag[x] = true;  QUEUE.pop();  if (QUEUE.size() == 0) break;  }  cout << "Кратчайшие пути:\n";  for (int i = 0; i < len; i++)  {  if (result[i] == 10000) cout << i + 1 << " - недоступен\n";  else cout << i + 1 << " - " << result[i] << endl;  }   system("pause"); } |

**Результаты выполнения**



**Реализация на Qt**

Файлы проекта на Qt:



Вывод программы на Qt:

