|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** \_***ИУК «Информатика и управление»*\_\_**\_\_\_\_\_\_\_\_\_\_\_\_

**КАФЕДРА** \_\_***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

**ДОМАШНЯЯ РАБОТА №2**

**«Решение задач оптимизации при принятии решений»**

**ДИСЦИПЛИНА: «Типы и структуры данных»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-32Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_Зудин Д.В.\_\_\_\_\_)  (Подпись) (Ф.И.О.) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_Пчелинцева Н.И.\_\_\_)  (Подпись) (Ф.И.О.) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |
| Калуга, 2022 г.  **Цель**: формирование практических навыков создания алгоритмов решения оптимизационных задач.  **Задачи**:  1. Изучить виды задач оптимизации при принятии решений;  2. Изучить основные алгоритмы для решения данных задач;  3. Реализовать алгоритм согласно варианту.  **Вариант №4**  **Формулировка задания**  1. Разработать консольное приложение, написанное с помощью объектно-ориентированной технологии. Индивидуальное задание предусмотрено вариантом, который назначает преподаватель.  2. Приложение необходимо запускать для демонстрации из командной строки с указанием названий приложения и трех файлов:  − все входные данные (например, последовательности чисел, коэффициенты многочленов и т.д.) считать из первого файла;  − все выходные данные записать во второй файл;  − все возникшие ошибки записать в третий файл – файл ошибок.  3. Все основные сущности приложения представить в виде отдельных классов.  4. Необходимо предусмотреть пользовательское меню, содержащее набор команд всех основных операций для работы с графом, а также команду для запуска индивидуального задания.  5. В приложении также должны быть учтены все критические ситуации, обработанные с помощью класса исключений.  **Индивидуальное задание**  Реализовать алгоритм поиска максимального потока с одним истоком и одним стоком на основе алгоритма поиска в ширину.  **Листинг файла ConsoleEditor.h**  #pragma once  #include <Windows.h>  inline void setConsoleColor(int color = 7)  {  HANDLE handle = GetStdHandle(STD\_OUTPUT\_HANDLE);  SetConsoleTextAttribute(handle, color);  }  inline void setConsoleCoordinates(int x, int y)  {  COORD position = { x, y };  HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);  SetConsoleCursorPosition(hConsole, position);  }  inline void HideCursor()  {  HANDLE handle = GetStdHandle(STD\_OUTPUT\_HANDLE);  CONSOLE\_CURSOR\_INFO structCursorInfo;  GetConsoleCursorInfo(handle, &structCursorInfo);  structCursorInfo.bVisible = FALSE;  SetConsoleCursorInfo(handle, &structCursorInfo);  }  inline void ShowCursor()  {  HANDLE handle = GetStdHandle(STD\_OUTPUT\_HANDLE);  CONSOLE\_CURSOR\_INFO structCursorInfo;  GetConsoleCursorInfo(handle, &structCursorInfo);  structCursorInfo.bVisible = TRUE;  SetConsoleCursorInfo(handle, &structCursorInfo);  }  inline int getConsoleWidth()  {  HANDLE handle = GetStdHandle(STD\_OUTPUT\_HANDLE);  CONSOLE\_SCREEN\_BUFFER\_INFO consoleInfo;  if (GetConsoleScreenBufferInfo(handle, &consoleInfo))  {  return consoleInfo.srWindow.Right - consoleInfo.srWindow.Left + 1;  }  return 0;  }  inline int getConsoleHeight()  {  HANDLE handle = GetStdHandle(STD\_OUTPUT\_HANDLE);  CONSOLE\_SCREEN\_BUFFER\_INFO consoleInfo;  if (GetConsoleScreenBufferInfo(handle, &consoleInfo))  {  return consoleInfo.srWindow.Bottom - consoleInfo.srWindow.Top + 1;  }  return 0;  }  **Листинг файла Menu.h**  #ifndef MENU  #define MENU  #include <iostream>  #include <string>  #include <any>  #include <vector>  #include <functional>  #include <conio.h>  #include "ConsoleEditor.h"  class Menu  {  public:  using Func = std::function<void(std::vector<std::any>)>;  Menu(std::string label, Func function);  Menu(std::string label, std::vector<Menu> menus);  Menu(std::string label, Func function, std::vector<std::any> params);  Menu(std::string label, std::vector<Menu> menus, std::vector<std::any> params);  Menu(const Menu& menu);  void Run(std::vector<std::any> params);  void PrintMenu(size\_t selected = 1);  void changeMenu(size\_t old\_selected, size\_t new\_selected);  void setSelectedItemColor(int selectedItemColor);  private:  enum Buttons  {  ARROW = 224,  UP = 80,  DOWN = 72,  ESC = 27,  ENTER = 13  };  int selectedItemColor = BACKGROUND\_GREEN;  std::string label{};  std::vector<Menu> menus{};  Func func{};  std::vector<std::any> params{};  };  #endif  **Листинг файла Menu.cpp**  #include "Menu.h"  Menu::Menu(std::string label, Func func) :  label(label), func(func)  {}  Menu::Menu(std::string label, std::vector<Menu> menus) :  label(label), menus(menus)  {}  Menu::Menu(std::string label, Func func, std::vector<std::any> params) :  label(label), func(func), params(params)  {}  Menu::Menu(std::string label, std::vector<Menu> menus, std::vector<std::any> params) :  label(label), menus(menus), params(params)  {}  Menu::Menu(const Menu& menu)  {  label = menu.label;  menus = menu.menus;  func = menu.func;  params = menu.params;  }  void Menu::setSelectedItemColor(int selectedItemColor)  {  this->selectedItemColor = selectedItemColor;  }  void Menu::PrintMenu(size\_t selected)  {  HideCursor();  if (!func)  {  std::cout << "Меню '" << label << "'\n";  }  for (size\_t i = 0; i < menus.size(); i++)  {  if (i + 1 == selected)  {  setConsoleColor(selectedItemColor);  }  std::cout << i + 1 << ". " << menus[i].label;  setConsoleColor();  std::cout << std::endl;  }  if (!func)  {  std::cout << "Нажмите ESC для выхода из '" << label << "'\n";  }  }  // для оптимизации отрисовки меню  void Menu::changeMenu(size\_t old\_selected, size\_t new\_selected)  {  HideCursor();  setConsoleCoordinates(0, old\_selected);  std::cout << old\_selected << ". " << menus[old\_selected - 1].label;  setConsoleCoordinates(0, new\_selected);  setConsoleColor(selectedItemColor);  std::cout << new\_selected << ". " << menus[new\_selected - 1].label;  setConsoleColor();  }  void Menu::Run(std::vector<std::any> params)  {  bool exit = false;  while (!exit)  {  system("cls");  PrintMenu();  int select = 1;  if (!func)  {  int key = 0;  int old\_select = 0;  while (key != ENTER && key != ESC)  {  int oldWidth = getConsoleWidth();  int oldHeight = getConsoleHeight();  // Нажатие на стрелку вверх(вниз) генерирует два события  // с кодом ARROW=224 и с кодом UP=80(DOWN=72)  key = ARROW;  while (key == ARROW)  {  while (!\_kbhit())  {  int newWidth = getConsoleWidth();  int newHeight = getConsoleHeight();  // Если размеры консоли изменяются,  // то перерисовываем,  // иначе будет цветная полоса во всю строку  if (oldWidth != newWidth || oldHeight != newHeight)  {  system("cls");  PrintMenu(select);  oldWidth = newWidth;  oldHeight = newHeight;  }  }  key = \_getch();  }  old\_select = select;  switch (key)  {  case UP:  select = (select >= menus.size() ? 1 : select + 1);  changeMenu(old\_select, select);  break;  case DOWN:  select = (select <= 1 ? menus.size() : select - 1);  changeMenu(old\_select, select);  break;  case ESC:  select = 0;  exit = true;  setConsoleCoordinates(0, menus.size() + 2);  break;  default:  break;  }  }  if (select == 0)  {  exit = true;  }  else  {  menus[select - 1].Run(params);  }  }  else  {  func(params);  system("pause");  exit = true;  }  }  }  **Листинг файла FileLogging.h**  #ifndef FILE\_LOGGING  #define FILE\_LOGGING  #include <string>  #include <fstream>  #include <ctime>  #include <iostream>  class FileLogging  {  public:  FileLogging(std::string fileName);  void Logging(std::string message);  private:  std::string getTime();  std::string fileName;  };  #endif  **Листинг файла FileLogging.cpp**  #define \_CRT\_SECURE\_NO\_WARNINGS  #include "FileLogging.h"  FileLogging::FileLogging(std::string fileName)  {  this->fileName = fileName;  }  void FileLogging::Logging(std::string message)  {  std::ofstream fout(fileName, std::ios::out | std::ios::app);  if (fout.is\_open())  {  fout << "[" << getTime() << "]" << message << "\n";  }  fout.close();  }  std::string FileLogging::getTime()  {  time\_t seconds = time(nullptr);  tm\* timeinfo = localtime(&seconds);  std::string currTime = asctime(timeinfo);  currTime.pop\_back();  return currTime;  }  **Листинг файла HelpFunctions.h**  #ifndef HELP\_FUNCTIONS  #define HELP\_FUNCTIONS  #include <random>  #include <ctime>  #include <climits>  #include <algorithm>  #include <vector>  #include <iostream>  #include <exception>  #include <iterator>  #include <sstream>  #include "FileLogging.h"  inline int getRandom(const int \_min, const int \_max)  {  return rand() % (\_max - \_min + 1) + \_min;  }  inline double inf()  {  return std::numeric\_limits<double>::infinity();  }  inline double getMatrixMaximum(const std::vector<std::vector<double>>& matrix)  {  double \_max = -inf();  for (auto i : matrix)  {  \_max = std::max(\*std::max\_element(i.begin(), i.end()), \_max);  }  return \_max;  }  template <typename T>  std::ostream& operator<<(std::ostream& out, const std::vector<T>& v)  {  out << "[";  for (size\_t i = 0; i < v.size(); i++)  {  out << v[i] << (i == v.size() - 1 ? "" : ", ");  }  out << "]";  return out;  }  // std::vector<T> -> std::string  template <typename T>  inline std::string vtos(std::vector<T>& v)  {  std::stringstream ss;  ss << v;  return ss.str();  }  //std::vector<std::vector<T>> -> std::string  template <typename T>  inline std::string vvtos(std::vector<std::vector<T>>& vv)  {  std::stringstream ss;  for (auto v : vv)  {  ss << v << "\n";  }  return ss.str();  }  inline int InputInt(const std::string MSG, const int MIN, const int MAX)  {  int input{};  bool exit = false;  while (!exit)  {  std::cout << MSG;  std::string strInput;  getline(std::cin, strInput);  try  {  // Проверка strInput на наличие лишних символов (не цифр)  // и выброс исключения std::invalid\_argument,  // иначе можно ввести такие strInput, что они начинаются с цифр  // и заканчиваются другими символами  for (size\_t i = 0; i < strInput.length(); i++)  {  if (strInput[i] == '-' && i == 0)  {  continue;  }  if (strInput[i] < '0' || strInput[i] > '9')  {  throw std::invalid\_argument("You can enter integer numbers only.");  }  }  input = std::stoi(strInput);  exit = true;  }  catch (std::invalid\_argument const&)  {  std::cout << "Можно ввести только целое число!\n";  }  catch (std::out\_of\_range const&)  {  std::cout << "Введенное число выходит из допустимого диапазона!\n";  }  catch (...)  {  std::cout << "Неизвестная ошибка при вводе!\n";  }  if (exit && (input < MIN || input > MAX))  {  std::cout << "Введенное число выходит из допустимого диапазона!\n";  exit = false;  }  }  return input;  }  inline int InputInt(const std::string MSG, const int MIN, const int MAX, FileLogging\* flog)  {  int input{};  bool exit = false;  while (!exit)  {  std::cout << MSG;  std::string strInput;  getline(std::cin, strInput);  try  {  // Проверка strInput на наличие лишних символов (не цифр)  // и выброс исключения std::invalid\_argument,  // иначе можно ввести такие strInput, что они начинаются с цифр  // и заканчиваются другими символами  for (size\_t i = 0; i < strInput.length(); i++)  {  if (strInput[i] == '-' && i == 0)  {  continue;  }  if (strInput[i] < '0' || strInput[i] > '9')  {  throw std::invalid\_argument("You can enter integer numbers only.");  if (flog)  {  flog->Logging("Incorrect number entry");  }  }  }  input = std::stoi(strInput);  exit = true;  }  catch (std::invalid\_argument const&)  {  std::cout << "Можно ввести только целое число!\n";  if (flog)  {  flog->Logging("Incorrect number entry");  }  }  catch (std::out\_of\_range const&)  {  std::cout << "Введенное число выходит из допустимого диапазона!\n";  if (flog)  {  flog->Logging("The entered number out of range");  }  }  catch (...)  {  std::cout << "Неизвестная ошибка при вводе!\n";  if (flog)  {  flog->Logging("Unknown input error");  }  }  if (exit && (input < MIN || input > MAX))  {  std::cout << "Введенное число выходит из допустимого диапазона!\n";  exit = false;  if (flog)  {  flog->Logging("The entered number out of range");  }  }  }  return input;  }  inline std::string concat(std::string s1, std::string s2)  {  std::stringstream ss;  ss << s1 << s2;  return ss.str();  }  #endif  **Листинг файла Graph.h**  #ifndef GRAPH  #define GRAPH  #include <vector>  #include <iostream>  #include <exception>  #include <stack>  #include <queue>  #include <iomanip>  #include <sstream>  #include <numeric>  #include "HelpFunctions.h"  class Graph  {  public:  // Матрица весов  using WeightMatrix = std::vector<std::vector<double>>;  Graph(const Graph& graph);  Graph(const WeightMatrix& weightMatrix);  Graph(const size\_t n);  WeightMatrix& getWeightMatrix();  void ClearGraph();  void FillRandomly(const int MIN\_WEIGHT = 1, const int MAX\_WEIGHT = 1);  size\_t getVertexCount() const;  void InsertVertex();  void DeleteVertex(const size\_t i);  // Создать дугу  void CreateArc(const size\_t i, const size\_t j, const double WEIGHT = 1);  // Создать ребро  void CreateEdge(const size\_t i, const size\_t j, const double WEIGHT = 1);  // Удалить дугу  void DeleteArc(const size\_t i, const size\_t j);  // Удалить ребро  void DeleteEdge(const size\_t i, const size\_t j);  // Вывод матрицы весов с точностью весов PRECISION  void PrintGraph(const size\_t PRECISION = 0) const;  // Обход в глубину, начиная с вершины i (возвращает путь)  std::vector<size\_t> DepthFirstSearch(size\_t i = 0) const;  // Обход в ширину, начиная с вершины i (возвращает путь)  std::vector<size\_t> BreadthFirstSearch(size\_t i = 0) const;  // Алгоритм Дейкстры, возвращающий пару векторов: вектор кратчайших путей(веса)  // и вектор путей  std::pair<std::vector<double>, std::vector<std::vector<size\_t>>> Dijkstra(const size\_t STARTING\_VERTEX = 0) const;  // Возвращает вектор гамильтоновых путей графа - простых путей (т.е. без петель),  // проходящих через каждую вершину графа только один раз  std::vector<std::vector<size\_t>> HamiltonianPath() const;  // Возвращает максимальный поток из истока s в сток t  int FordFulkerson(const int s, const int t) const;  private:  WeightMatrix weightMatrix{};  // Существует ли связь между вершинами i и j  bool ExistLink(const size\_t i, const size\_t j) const;  int FordFulkerson(const int s, const int t, std::vector<std::vector<int>>& graph) const;  int BFSforFordFulkerson(const int s, const int t, std::vector<int>& parent, std::vector<std::vector<int>>& graph) const;  };  #endif  **Листинг файла Graph.cpp**  #include "Graph.h"  Graph::Graph(const Graph& graph) : weightMatrix(graph.weightMatrix) {}  Graph::Graph(const WeightMatrix& weightMatrix)  {  for (auto i : weightMatrix)  {  if (i.size() != weightMatrix.size())  {  throw std::invalid\_argument("Weight matrix must be square");  }  }  this->weightMatrix = weightMatrix;  }  Graph::Graph(const size\_t n) : weightMatrix(WeightMatrix(n, std::vector<double>(n, inf())))  {}  Graph::WeightMatrix& Graph::getWeightMatrix()  {  return weightMatrix;  }  std::vector<size\_t> Graph::DepthFirstSearch(size\_t i) const  {  std::vector<size\_t> path;  if (getVertexCount() == 1)  {  path.push\_back(0);  return path;  }  std::vector<bool> visited(getVertexCount());  std::stack<size\_t> currVertices;  currVertices.push(i);  while (!currVertices.empty())  {  // Берем вершину i из стека и помечаем её как пройденную  i = currVertices.top();  currVertices.pop();  if (!visited[i])  {  path.push\_back(i);  }  visited[i] = true;  for (size\_t j = getVertexCount() - 1; j > 0; j--)  {  // Если есть связь i -> j и j не посещена ранее,  if (ExistLink(i, j) && !visited[j])  {  // То добавляем ее в стек текущих вершин  currVertices.push(j);  }  }  }  return path;  }  std::vector<size\_t> Graph::BreadthFirstSearch(size\_t i) const  {  std::vector<size\_t> path;  if (getVertexCount() == 1)  {  path.push\_back(0);  return path;  }  std::queue<size\_t> currVertices;  std::vector<bool> visited(getVertexCount());  currVertices.push(i);  path.push\_back(i);  visited[i] = true;  while (!currVertices.empty())  {  i = currVertices.front();  currVertices.pop();  for (size\_t j = 0; j < getVertexCount(); j++)  {  if (ExistLink(i, j) && !visited[j])  {  if (!visited[j])  {  path.push\_back(j);  }  visited[j] = true;  currVertices.push(j);  }  }  }  return path;  }  size\_t Graph::getVertexCount() const  {  return weightMatrix.size();  }  void Graph::ClearGraph()  {  Graph::WeightMatrix weightMatrix{};  this->weightMatrix = weightMatrix;  }  void Graph::FillRandomly(const int MIN\_WEIGHT, const int MAX\_WEIGHT)  {  srand(time(nullptr));  for (auto& i : weightMatrix)  {  for (auto& j : i)  {  j = getRandom(MIN\_WEIGHT, MAX\_WEIGHT + 1);  j = (j == MAX\_WEIGHT + 1 ? inf() : j);  }  }  }  void Graph::InsertVertex()  {  for (auto& i : weightMatrix)  {  i.push\_back(inf());  }  weightMatrix.push\_back(std::vector<double>(weightMatrix.size() + 1, inf()));  }  void Graph::DeleteVertex(const size\_t i)  {  weightMatrix.erase(weightMatrix.begin() + i, weightMatrix.begin() + i + 1);  for (auto& j : weightMatrix)  {  j.erase(j.begin() + i, j.begin() + i + 1);  }  }  void Graph::CreateArc(const size\_t i, const size\_t j, const double WEIGHT)  {  weightMatrix[i][j] = WEIGHT;  }  void Graph::CreateEdge(const size\_t i, const size\_t j, const double WEIGHT)  {  weightMatrix[i][j] = weightMatrix[j][i] = WEIGHT;  }  void Graph::DeleteArc(const size\_t i, const size\_t j)  {  weightMatrix[i][j] = inf();  }  void Graph::DeleteEdge(const size\_t i, const size\_t j)  {  weightMatrix[i][j] = weightMatrix[j][i] = inf();  }  void Graph::PrintGraph(const size\_t PRECISION) const  {  const std::string INF = "inf";  // находим максимальное число (по длине строкового представления)  // в матрице weightMatrix  // меняем в ней inf на -inf  WeightMatrix newWeightMatrix = weightMatrix;  for (auto& i : newWeightMatrix)  {  std::replace(i.begin(), i.end(), inf(), -inf());  }  double maxWeight = getMatrixMaximum(newWeightMatrix);  std::stringstream ssMaxWeight;  ssMaxWeight << std::fixed << std::setprecision(PRECISION) << maxWeight;  size\_t lenMaxWeight = ssMaxWeight.str().length();  bool edgesArePositive = true; // все ребра - положительные числа  for (auto i : weightMatrix)  {  for (auto j : i)  {  if (j < 0)  {  edgesArePositive = false;  break;  }  }  if (!edgesArePositive)  {  break;  }  }  // Вычисляем количество ячеек cellsCount под каждое число матрицы weightMatrix,  // чтобы матрица вывелась ровно  size\_t cellsCount = std::max(lenMaxWeight, INF.length()) + 2;  for (auto i : weightMatrix)  {  for (auto j : i)  {  if (j == inf())  {  std::cout << std::setw(cellsCount) << (edgesArePositive ? "0" : INF);  }  else  {  std::cout << std::setw(cellsCount) << std::fixed << std::setprecision(PRECISION) << j;  }  }  std::cout << "\n";  }  }  bool Graph::ExistLink(const size\_t i, const size\_t j) const  {  return weightMatrix[i][j] != inf();  }  std::pair<std::vector<double>, std::vector<std::vector<size\_t>>> Graph::Dijkstra(const size\_t STARTING\_VERTEX) const  {  // метка для вершины, по которой проходятся второй раз или для paths, что означает,  // что из в эту вершину попасть нельзя  const size\_t MARK = getVertexCount() + 1;  std::vector<double> shortestPaths(getVertexCount(), inf());  // если paths[i] = j != MARK, то из вершины j можно попасть напрямую в вершину i  // если paths[i] = j = MARK, то в вершину i нельзя попасть напрямую  std::vector<size\_t> paths(getVertexCount());  std::vector<bool> traversed(getVertexCount()); // пройденные вершины  size\_t currVertex = STARTING\_VERTEX;  shortestPaths[currVertex] = 0;  bool exit = false;  while (!exit)  {  traversed[currVertex] = (currVertex != MARK);  for (size\_t i = 0; i < getVertexCount(); i++)  {  if (ExistLink(currVertex, i) && !traversed[i])  {  if (shortestPaths[currVertex] + weightMatrix[currVertex][i] < shortestPaths[i])  {  paths[i] = currVertex;  shortestPaths[i] = shortestPaths[currVertex] + weightMatrix[currVertex][i];  }  }  }  currVertex = MARK; // Помечаем вершину, чтобы отследить, изменилась ли она  double minPath = inf();  for (size\_t i = 0; i < getVertexCount(); i++)  {  if (!traversed[i])  {  if (shortestPaths[i] < minPath)  {  minPath = shortestPaths[i];  currVertex = i;  }  }  }  exit = (std::all\_of(traversed.begin(), traversed.end(), [](bool v) { return v; })) || (currVertex == MARK);  }  std::replace\_if(paths.begin(), paths.end(), [&traversed](size\_t i) { return !traversed[i]; }, MARK);  // Получаем пути  std::vector<std::vector<size\_t>> directPaths;  for (size\_t i = 0; i < getVertexCount(); i++)  {  size\_t v = i;  std::vector<size\_t> path{v};  while (v != STARTING\_VERTEX)  {  v = paths[v];  path.insert(path.begin(), v);  if (v == MARK)  {  path.clear();  break;  }  }  directPaths.push\_back(path);  }  return std::make\_pair(shortestPaths, directPaths);  }  std::vector<std::vector<size\_t>> Graph::HamiltonianPath() const  {  std::vector<std::vector<size\_t>> paths;  std::vector<size\_t> vertices(getVertexCount());  std::iota(vertices.begin(), vertices.end(), 0);  do  {  bool valid = true;  for (size\_t i = 0; i < vertices.size() - 1; i++)  {  if (!ExistLink(vertices[i], vertices[i + 1]))  {  valid = false;  break;  }  }  if (valid)  {  paths.push\_back(vertices);  }  } while (std::next\_permutation(vertices.begin(), vertices.end()));  return paths;  }  int Graph::BFSforFordFulkerson(const int s, const int target, std::vector<int>& parent, std::vector<std::vector<int>>& graph) const  {  fill(parent.begin(), parent.end(), -1);  parent[s] = -2;  std::queue<std::pair<int, int>> q;  q.push({ s, INT\_MAX });  while (!q.empty())  {  int u = q.front().first;  int cap = q.front().second;  q.pop();  for (int v = 0; v < getVertexCount(); v++)  {  if (u != v && graph[u][v] != 0 && parent[v] == -1)  {  parent[v] = u;  int min\_cap = std::min(cap, graph[u][v]);  if (v == target)  {  return min\_cap;  }  q.push({ v, min\_cap });  }  }  }  return 0;  }  int Graph::FordFulkerson(const int s, const int t) const  {  const int V = getVertexCount();  std::vector<std::vector<int>> rGraph(V, std::vector<int>(V)); // остаточные пропускные способности  for (int i = 0; i < V; i++)  {  for (int j = 0; j < V; j++)  {  rGraph[i][j] = (weightMatrix[i][j] == inf() ? 0 : weightMatrix[i][j]);  }  }  return FordFulkerson(s, t, rGraph);  }  int Graph::FordFulkerson(const int s, const int t, std::vector<std::vector<int>>& graph) const  {  std::vector<int> parent(getVertexCount(), -1);  int max\_flow = 0;  int min\_cap = 0;  while (min\_cap = BFSforFordFulkerson(s, t, parent, graph))  {  max\_flow += min\_cap;  int v = t;  while (v != s)  {  int u = parent[v];  graph[u][v] -= min\_cap;  graph[v][u] += min\_cap;  v = u;  }  }  return max\_flow;  }  **Листинг файла WorkWithGraph.h**  #ifndef WORK\_WITH\_GRAPH  #define WORK\_WITH\_GRAPH  #include "Graph.h"  #include "HelpFunctions.h"  #include "FileLogging.h"  #include "Menu.h"  inline void mPrintGraph(std::vector<std::any> params)  {  auto graph = std::any\_cast<Graph\*>(params[0]);  auto foutlog = std::any\_cast<FileLogging>(params[2]);  auto ferrlog = std::any\_cast<FileLogging>(params[3]);  foutlog.Logging(concat("Function call: ", \_\_func\_\_));  if (graph->getVertexCount() == 0)  {  std::cout << "Граф пустой!\n";  ferrlog.Logging("Attempt to display an empty graph");  return;  }  graph->PrintGraph();  foutlog.Logging(concat("\n", vvtos(graph->getWeightMatrix())));  }  inline void mGraphIsEmpty(std::vector<std::any> params)  {  auto graph = std::any\_cast<Graph\*>(params[0]);  auto foutlog = std::any\_cast<FileLogging>(params[2]);  foutlog.Logging(concat("Function call: ", \_\_func\_\_));  std::cout << "Граф пуст: " << (graph->getVertexCount() == 0 ? "да" : "нет") << "\n";  foutlog.Logging(concat("\n", vvtos(graph->getWeightMatrix())));  }  inline void mInsertVertex(std::vector<std::any> params)  {  auto graph = std::any\_cast<Graph\*>(params[0]);  auto foutlog = std::any\_cast<FileLogging>(params[2]);  auto ferrlog = std::any\_cast<FileLogging>(params[3]);  foutlog.Logging(concat("Function call: ", \_\_func\_\_));  if (graph->getVertexCount() == INT\_MAX)  {  std::cout << "Граф слишком большой!\n";  ferrlog.Logging("Attempt to add a vertex to a very large graph");  }  graph->InsertVertex();  std::cout << "Вершина успешно добавлена!";  foutlog.Logging(concat("\n", vvtos(graph->getWeightMatrix())));  }  inline void mDeleteVertex(std::vector<std::any> params)  {  auto graph = std::any\_cast<Graph\*>(params[0]);  auto foutlog = std::any\_cast<FileLogging>(params[2]);  auto ferrlog = std::any\_cast<FileLogging>(params[3]);  foutlog.Logging(concat("Function call: ", \_\_func\_\_));  if (graph->getVertexCount() == 0)  {  std::cout << "Граф пустой!\n";  ferrlog.Logging("Attempt to remove a vertex from an empty graph");  return;  }  int v = InputInt("Введите номер вершины, которую необходимо удалить: ", 0, graph->getVertexCount() - 1, &ferrlog);  graph->DeleteVertex(v);  std::cout << "Вершина " << v << " успешно удалена!\n";  foutlog.Logging(concat("\n", vvtos(graph->getWeightMatrix())));  }  inline void mCreateEdge(std::vector<std::any> params)  {  auto graph = std::any\_cast<Graph\*>(params[0]);  auto foutlog = std::any\_cast<FileLogging>(params[2]);  auto ferrlog = std::any\_cast<FileLogging>(params[3]);  foutlog.Logging(concat("Function call: ", \_\_func\_\_));  if (graph->getVertexCount() == 0)  {  std::cout << "Граф пустой!\n";  ferrlog.Logging("Attempt to create an edge in an empty graph");  return;  }  int v1 = InputInt("Введите номер первой вершины: ", 0, graph->getVertexCount() - 1, &ferrlog);  int v2 = InputInt("Введите номер второй вершины: ", 0, graph->getVertexCount() - 1, &ferrlog);  int w = InputInt("Введите вес ребра: ", INT\_MIN, INT\_MAX, &ferrlog);  graph->CreateEdge(v1, v2, w);  std::cout << "Ребро (" << v1 << ", " << v2 << ") успешно добавлено!\n";  foutlog.Logging(concat("\n", vvtos(graph->getWeightMatrix())));  }  inline void mCreateArc(std::vector<std::any> params)  {  auto graph = std::any\_cast<Graph\*>(params[0]);  auto foutlog = std::any\_cast<FileLogging>(params[2]);  auto ferrlog = std::any\_cast<FileLogging>(params[3]);  foutlog.Logging(concat("Function call: ", \_\_func\_\_));  if (graph->getVertexCount() == 0)  {  std::cout << "Граф пустой!\n";  ferrlog.Logging("Attempt to create an arc in an empty graph");  return;  }  int v1 = InputInt("Введите номер первой вершины: ", 0, graph->getVertexCount() - 1, &ferrlog);  int v2 = InputInt("Введите номер второй вершины: ", 0, graph->getVertexCount() - 1, &ferrlog);  int w = InputInt("Введите вес ребра: ", INT\_MIN, INT\_MAX, &ferrlog);  graph->CreateArc(v1, v2, w);  std::cout << "Дуга (" << v1 << ", " << v2 << ") успешно добавлено!\n";  foutlog.Logging(concat("\n", vvtos(graph->getWeightMatrix())));  }  inline void mDeleteEdge(std::vector<std::any> params)  {  auto graph = std::any\_cast<Graph\*>(params[0]);  auto foutlog = std::any\_cast<FileLogging>(params[2]);  auto ferrlog = std::any\_cast<FileLogging>(params[3]);  foutlog.Logging(concat("Function call: ", \_\_func\_\_));  if (graph->getVertexCount() == 0)  {  std::cout << "Граф пустой!\n";  ferrlog.Logging("Attempt to remove an edge from an empty graph");  return;  }  int v1 = InputInt("Введите номер первой вершины: ", 0, graph->getVertexCount() - 1, &ferrlog);  int v2 = InputInt("Введите номер второй вершины: ", 0, graph->getVertexCount() - 1, &ferrlog);  graph->DeleteEdge(v1, v2);  std::cout << "Ребро (" << v1 << ", " << v2 << ") успешно удалено!\n";  foutlog.Logging(concat("\n", vvtos(graph->getWeightMatrix())));  }  inline void mDeleteArc(std::vector<std::any> params)  {  auto graph = std::any\_cast<Graph\*>(params[0]);  auto foutlog = std::any\_cast<FileLogging>(params[2]);  auto ferrlog = std::any\_cast<FileLogging>(params[3]);  foutlog.Logging(concat("Function call: ", \_\_func\_\_));  if (graph->getVertexCount() == 0)  {  std::cout << "Граф пустой!\n";  ferrlog.Logging("Attempt to remove an arc from an empty graph");  return;  }  int v1 = InputInt("Введите номер первой вершины: ", 0, graph->getVertexCount() - 1, &ferrlog);  int v2 = InputInt("Введите номер второй вершины: ", 0, graph->getVertexCount() - 1, &ferrlog);  graph->DeleteArc(v1, v2);  std::cout << "Дуга (" << v1 << ", " << v2 << ") успешно удалено!\n";  foutlog.Logging(concat("\n", vvtos(graph->getWeightMatrix())));  }  inline void mFillRandomly(std::vector<std::any> params)  {  auto graph = std::any\_cast<Graph\*>(params[0]);  auto foutlog = std::any\_cast<FileLogging>(params[2]);  auto ferrlog = std::any\_cast<FileLogging>(params[3]);  foutlog.Logging(concat("Function call: ", \_\_func\_\_));  if (graph->getVertexCount() == 0)  {  std::cout << "Граф пустой!\n";  ferrlog.Logging("Attempt to fill an empty graph with random numbers");  return;  }  int minw = InputInt("Введите минимальный вес: ", INT\_MIN, INT\_MAX, &ferrlog);  int maxw = InputInt("Введите максимальный вес: ", INT\_MIN, INT\_MAX, &ferrlog);  graph->FillRandomly(minw, maxw);  std::cout << "Матрица весов заполнена случайными числами!\n";  foutlog.Logging(concat("\n", vvtos(graph->getWeightMatrix())));  }  inline void mBreadthFirstSearch(std::vector<std::any> params)  {  auto graph = std::any\_cast<Graph\*>(params[0]);  auto foutlog = std::any\_cast<FileLogging>(params[2]);  auto ferrlog = std::any\_cast<FileLogging>(params[3]);  foutlog.Logging(concat("Function call: ", \_\_func\_\_));  if (graph->getVertexCount() == 0)  {  std::cout << "Граф пустой!\n";  ferrlog.Logging("Attempt to apply a BFS in an empty graph");  return;  }  int v = InputInt("Введите номер вершины, с которой надо начинать обход: ", 0, graph->getVertexCount() - 1, &ferrlog);  std::cout << graph->BreadthFirstSearch(v) << "\n";  foutlog.Logging(concat("\n", vvtos(graph->getWeightMatrix())));  }  inline void mDepthFirstSearch(std::vector<std::any> params)  {  auto graph = std::any\_cast<Graph\*>(params[0]);  auto foutlog = std::any\_cast<FileLogging>(params[2]);  auto ferrlog = std::any\_cast<FileLogging>(params[3]);  foutlog.Logging(concat("Function call: ", \_\_func\_\_));  if (graph->getVertexCount() == 0)  {  std::cout << "Граф пустой!\n";  ferrlog.Logging("Attempt to apply a DFS in an empty graph");  return;  }  int v = InputInt("Введите номер вершины, с которой надо начинать обход: ", 0, graph->getVertexCount() - 1, &ferrlog);  std::cout << graph->DepthFirstSearch(v) << "\n";  foutlog.Logging(concat("\n", vvtos(graph->getWeightMatrix())));  }  inline void mClearGraph(std::vector<std::any> params)  {  auto graph = std::any\_cast<Graph\*>(params[0]);  auto foutlog = std::any\_cast<FileLogging>(params[2]);  auto ferrlog = std::any\_cast<FileLogging>(params[3]);  foutlog.Logging(concat("Function call: ", \_\_func\_\_));  if (graph->getVertexCount() == 0)  {  std::cout << "Граф пустой!\n";  ferrlog.Logging("Attempt to clear an empty graph");  return;  }  graph->ClearGraph();  std::cout << "Теперь граф пуст!\n";  foutlog.Logging(concat("\n", vvtos(graph->getWeightMatrix())));  }  inline void mDijkstra(std::vector<std::any> params)  {  auto graph = std::any\_cast<Graph\*>(params[0]);  auto foutlog = std::any\_cast<FileLogging>(params[2]);  auto ferrlog = std::any\_cast<FileLogging>(params[3]);  foutlog.Logging(concat("Function call: ", \_\_func\_\_));  if (graph->getVertexCount() == 0)  {  std::cout << "Граф пустой!\n";  ferrlog.Logging("Attempt to apply Dijkstra's algorithm to an empty graph");  return;  }  int v = InputInt("Введите начальную вершину: ", 0, graph->getVertexCount() - 1, &ferrlog);  auto dijksta = graph->Dijkstra(v);  for (size\_t i = 0; i < dijksta.first.size(); i++)  {  if (v != int(i))  {  std::cout << "Путь " << v << "->" << i << ": " << dijksta.second[i] << " Длина пути: " << dijksta.first[i] << "\n";  }  }  foutlog.Logging(concat("\n", vvtos(graph->getWeightMatrix())));  }  inline void mHamiltonianPath(std::vector<std::any> params)  {  auto graph = std::any\_cast<Graph\*>(params[0]);  auto foutlog = std::any\_cast<FileLogging>(params[2]);  auto ferrlog = std::any\_cast<FileLogging>(params[3]);  foutlog.Logging(concat("Function call: ", \_\_func\_\_));  if (graph->getVertexCount() == 0)  {  std::cout << "Граф пустой!\n";  ferrlog.Logging("Attempt to find Hamiltonian paths in an empty graph");  return;  }  auto hps = graph->HamiltonianPath();  std::cout << "Гамильтоновы пути:\n";  if (hps.empty())  {  std::cout << "Не существуют в данном графе!\n";  }  else  {  for (auto i : hps)  {  std::cout << i << "\n";  }  }  foutlog.Logging(concat("\n", vvtos(graph->getWeightMatrix())));  }  inline void mFordFulkerson(std::vector<std::any> params)  {  auto graph = std::any\_cast<Graph\*>(params[0]);  auto foutlog = std::any\_cast<FileLogging>(params[2]);  auto ferrlog = std::any\_cast<FileLogging>(params[3]);  foutlog.Logging(concat("Function call: ", \_\_func\_\_));  if (graph->getVertexCount() == 0)  {  std::cout << "Граф пустой!\n";  ferrlog.Logging("Attempt to apply Dijkstra's algorithm to an empty graph");  return;  }  int s = InputInt("Введите вершину-исток: ", 0, graph->getVertexCount() - 1, &ferrlog);  int t = InputInt("Введите вершину-сток: ", 0, graph->getVertexCount() - 1, &ferrlog);  if (s == t)  {  std::cout << "Исток и сток должны различаться!\n";  ferrlog.Logging("Source and sink are equal");  }  std::cout << "Максимальный поток равен " << graph->FordFulkerson(s, t) << "\n";  foutlog.Logging(concat("\n", vvtos(graph->getWeightMatrix())));  }  inline void mReadGraphFromFile(std::vector<std::any> params)  {  auto graph = std::any\_cast<Graph\*>(params[0]);  auto finpdata = std::any\_cast<std::string>(params[1]);  auto foutlog = std::any\_cast<FileLogging>(params[2]);  auto ferrlog = std::any\_cast<FileLogging>(params[3]);  foutlog.Logging(concat("Function call: ", \_\_func\_\_));  std::fstream fin(finpdata, std::ios::in);  Graph::WeightMatrix weightMatrix;  std::vector<double> buffer;  if (fin.is\_open())  {  std::string data;  while (fin >> data)  {  buffer.push\_back(data == "inf" ? inf() : std::stoi(data));  }  int wmsize = std::sqrt(buffer.size());  if (wmsize \* wmsize == buffer.size())  {  for (int i = 0; i < wmsize; i++)  {  std::vector<double> v;  for (int j = 0; j < wmsize; j++)  {  v.push\_back(buffer[i \* wmsize + j]);  }  weightMatrix.push\_back(v);  }  Graph newGraph(weightMatrix);  \*graph = newGraph;  std::cout << "Граф успешно считан из файла " << finpdata << "!\n";  }  else  {  std::cout << "Матрица весов в файле не квадратная!\n";  ferrlog.Logging("Weight matrix is not square");  }  }  else  {  std::cout << "Ошибка при открытии файла!\n";  ferrlog.Logging("Error opening file");  }  foutlog.Logging(concat("\n", vvtos(graph->getWeightMatrix())));  }  #endif  **Листинг файла main.cpp**  #include <iostream>  #include "WorkWithGraph.h"  int main(int argc, char\* argv[])  {  setlocale(LC\_ALL, "Russian");  std::string file\_input\_data = "input\_data.txt";  std::string file\_output\_log = "output\_log.txt";  std::string file\_error\_log = "error\_log.txt";  if (argc >= 3)  {  file\_input\_data = argv[1];  file\_output\_log = argv[2];  file\_error\_log = argv[3];  }  FileLogging error\_log(file\_error\_log);  FileLogging output\_log(file\_output\_log);  Graph::WeightMatrix weightMatrix{};  Graph\* graph = new Graph(weightMatrix);  std::vector<std::any> params{ graph, file\_input\_data, output\_log, error\_log };  Menu menu = Menu("Главное", std::vector<Menu>  {  Menu("Вывести граф (матрицу весов)", mPrintGraph),  Menu("Операции над графом", std::vector<Menu>  {  Menu("Заполнить матрицу весов случайными числами", mFillRandomly),  Menu("Очистить граф", mClearGraph),  Menu("Проверка графа на пустоту", mGraphIsEmpty),  Menu("Добавить вершину в граф", mInsertVertex),  Menu("Удалить вершину из графа", mDeleteVertex),  Menu("Создать ребро в графе", mCreateEdge),  Menu("Создать дугу в графе", mCreateArc),  Menu("Удалить ребро из графа", mDeleteEdge),  Menu("Удалить дугу из графа", mDeleteArc)  }),  Menu("Алгоритмы на графе", std::vector<Menu>  {  Menu("Обход графа", std::vector<Menu>  {  Menu("Обход в ширину", mBreadthFirstSearch),  Menu("Обход в глубину", mDepthFirstSearch)  }),  Menu("Поиск гамильтоновых путей", mHamiltonianPath),  Menu("Поиск кратчайшего пути Алгоритмом Дейкстры", mDijkstra),  Menu("Поиск максимального потока на основе поиска в ширину", mFordFulkerson)  }),  Menu("Считать граф из файла ", mReadGraphFromFile)  }, params);  menu.Run(params);  delete graph;  return 0;  }  **Результат выполнения программы**  **Главное меню**    **Вывод графа (матрицы пропускной способности)**    **Меню алгоритмов на графе**    **Поиск максимального потока**    **Выводы:**  В ходе работы были сформированы практические навыки создания алгоритмов решения оптимизационных задач. | | |
|  | | |