|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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 № 6** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Основные алгоритмы работы с графами.»** | |
|  | |
| Выполнил студент группы ИКБО-33-21 | Зарожина Я.А. |
| Принял преподаватель | Муравьева Е.А. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2022

# **Цель работы**

Получение практических навыков по выполнению операций над структурой данных граф.

# **Постановка задачи**

1. Разработать класс «Граф», обеспечивающий хранение и работу со структурой данных «граф», в соответствии с вариантом индивидуального задания. Реализовать метод ввода графа с клавиатуры, наполнение графа осуществлять с помощью метода добавления одного ребра. Реализовать метод вывода графа и методы, выполняющие задачи, определенные вариантом индивидуального задания.
2. Разработать программу, демонстрирующую работу всех методов класса.
3. Произвести тестирование программы на графе, предложенном в таблице 2.
4. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Условие задания:

|  |  |
| --- | --- |
| Представление графа в памяти | Задачи |
| Список смежных вершин | Составить программу нахождения кратчайших путей методом «Флойда».  Используя результат алгоритма вывести путь между вводимыми парами вершин. |

# **Решение**

Граф – это совокупность вершин, соединенных линиями (рёбрами).

Граф позволяет хранить и выводить данные в наглядном для пользователя виде.

Графы можно классифицировать на связные (если существует хотя бы один путь) и несвязные (если существует хотя бы одна несоединённая вершина); а также на ориентированные (если ребра имеют направление), неориентированные (если ребра не имеют направлений) и смешанные (когда есть и то и то). Также граф может быть взвешенным (если ребро имеет «вес») и невзвешенные (если у ребра отсутствует «вес»). Весом ребра называют расстояние от одной вершины до другой.

В практике часто нужно найти кратчайшее расстояние от одной вершины до другой. С этим может помочь алгоритм «Флойда-Уоршелла», который нацелен на нахождение кратчайших расстояний между всеми вершинами взвешенного [графа](http://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)).

Структура графа представлена в виде списка смежных вершин. Для этого внутри класса графа была создана структура узла, внутри которой хранилась структура списка взаимосвязи отношений вершин. Структура узла, в котором хранится список взаимосвязей, а также метод добавления нового элемента в список:

|  |
| --- |
| struct Node  {  struct List  {  List\* next = NULL;  int weight = 0;  int number = 0;  };  List\* Head = NULL;  List\* Last = NULL;  void AddNew(int weight, int number)  {  List\* list = new List;  list->weight = weight;  list->number = number;  if (!Head)  {  Head = list;  Last = list;  }  else  {  Last->next = list;  Last = list;  }  } |

Далее был создан конструктор класса графа, в котором и заполняется основная структура графа. Сам конструктор графа:

|  |
| --- |
| Graf()  {  int weight;  cout << "Введите кол-во вершин: ";  cin >> n;  matr = new Node[n];  for (int i = 0; i < n; i++)  {  cout << "\nВершина " << i+1 << ": ";  for (int j = 0; j < n; j++)  {  cin >> weight;  matr[i].AddNew(weight, j+1);  }  }  } |

Затем был создан метод вывода графа на экран, для этого была создана также вспомогательный метод в структуре узла, который проходился по всем элементам списка. Метод печати графа:

|  |
| --- |
| void GetGraf()  {  for (int i = 0; i < n; i++)  {  cout << "Вершина " << i + 1 << " имеет путь в: ";  matr[i].PrintList();  cout << "\n";  }  } |

Вспомогательный метод для прохождения по всему списку узла:

|  |
| --- |
| void PrintList()  {  int c = 0;  List\* list;  for (list = Head; list->next; list = list->next)  {  if (list->weight != 0)  {  cout << "Вершину " << list->number << " с весом " << list->weight << " ";  c++;  }  }  if (list->weight != 0)  {  cout << "Вершину " << list->number << " с весом " << list->weight;  c++;  }    if (c == 0)  cout << "Никуда";  } |

Был создан метод добавления в граф новой взаимосвязи между двумя существующими вершинами. В зависимости от ориентированности графа выбирался способ создания ребра. Там также использовался вспомогательный метод узла, который позволял изменять элементы в списке. Сам метод добавления ребра:

|  |
| --- |
| void AddEdge(int from, int to, int p, bool f)  {  if (f)  {  matr[from-1].ChangeValue(to-1, p);  }  else  {  matr[from - 1].ChangeValue(to - 1, p);  matr[to - 1].ChangeValue(from - 1, p);  }  } |

Вспомогательный метод, позволяющий изменять элементы в списке узла:

|  |
| --- |
| void ChangeValue(int to, int p)  {  List\* list = Head;  for (int i = 0; i != to; i++)  {  if (list->next)  list = list->next;  else  {  cout << "Введена неверная конечная вершина";  return;  }  }  list->weight = p;  } |

Метод Флойда, который выводит все кратчайшие пути до точек, если такие есть. В зависимости от введённого пользователем задания, выводит только короткие пути между всеми вершинами или выводит все короткие пути, а также кратчайшее расстояние и вершины, через которое это кратчайшее расстояние проходит, между введёнными пользователем вершинами. Для последнего используется вспомогательный рекурсивный метод вывода вершин, через который кратчайший путь лежит. Сам метод Флойда:

|  |
| --- |
| void Floyd(bool f, int from, int to)  {  Node::List\* l;  int i, j;  int\*\* a = new int\* [n+1];  if (f)  {  int\*\* b = new int\* [n+1];  for (i = 1; i <= n; i++)  {  a[i] = new int[n+1];  b[i] = new int[n+1];  }  for (i = 1; i <= n; i++)  for (j = 1; j <= n; j++)  {  a[i][j] = 0;  b[i][j] = 0;  }  for (i = 1; i <= n; i++)  {  l = matr[i-1].Head;  j = 1;  while (true)  {  if ((i != j) && (l->weight == 0))  a[i][j] = maxint;  else  a[i][j] = l->weight;  if (l->next)  l = l->next;  else  break;  j++;  }  }  for (int k = 1; k <= n; k++)  for (i = 1; i <= n; i++)  for (j = 1; j <= n; j++)  {  if (a[i][k] + a[k][j] < a[i][j])  {  a[i][j] = a[i][k] + a[k][j];  b[i][j] = k;  }  }  cout << "Кратчайшие пути:\n";  for (i = 1; i <= n; i++)  {  for (j = 1; j <= n; j++)  {  if (a[i][j] == maxint)  cout << "inf" << " ";  else  cout << a[i][j] << " ";  }  cout << "\n";  }  if (a[from][to] != maxint)  {  if (b[from][to] != 0)  {  cout << "Проходя через вершины: ";  NameH(b, from, to);  delete[] b;  }  else  cout << "Эти вершины соседи";  cout << " c длинной пути " << a[from][to];  }  else  cout << "Пути не существует";  }  else  {  for (i = 0; i < n; i++)  a[i] = new int[n];  for (i = 0; i < n; i++)  for (j = 0; j < n; j++)  a[i][j] = 0;  for (i = 0; i < n; i++)  {  l = matr[i].Head;  j = 0;  while (true)  {  a[i][j] = l->weight;  if (l->next)  l = l->next;  else  break;  j++;  }  }  for (int k = 0; k < n; k++)  for (i = 0; i < n; i++)  for (j = 0; j < n; j++)  if ((a[i][k] + a[k][j] < a[i][j]) || ((i != j) && (a[i][j] == 0)))  a[i][j] = a[i][k] + a[k][j];  cout << "Кратчайшие пути:\n";  for (i = 0; i < n; i++)  {  for (j = 0; j < n; j++)  cout << a[i][j] << " ";  cout << "\n";  }  }  delete[] a;  } |

Рекурсивный метод вывода вершин, находящихся на кратчайшем маршруте между введёнными пользователем вершинами:

|  |
| --- |
| void NameH(int\*\* b, int from, int to)  {  int k;  k = b[from][to];  if (k == 0)  return;  NameH(b, from, k);  cout << k<<" ";  NameH(b, k, to);  } |

При первом заходе в программу, пользователь види1т приглашение программы для ввода количества вершин (Рис. 1).



Рис. 1 - Первый вход в программу, ввод кол-ва вершин

При вводе количества вершин, программа просит сразу ввести первоначальные пути между графами для различных вершин (Рис. 2).

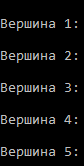


Рис. 2 - Первый вход в программу, ввод веса ребер между вершинами

После ввода данных программа выводит на экран свой основной интерфейс, где пользователю уже предлагается выбрать задание (Рис. 3).

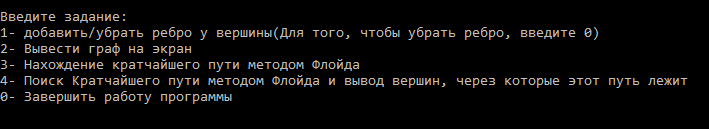


Рис. 3 - Интерфейс программы

# **Тестирование**

Первый тест проведём на неориентированном графе №1, удалив из него пару рёбер, что приведёт его к новому виду (Рис. 4).

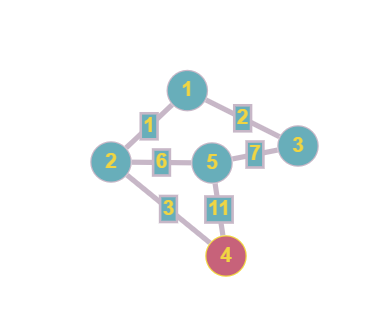


Рис. 4 - Граф первого теста

Для этого введём в программу первоначально координаты этого графа и выведем его на экран (Рис. 5).

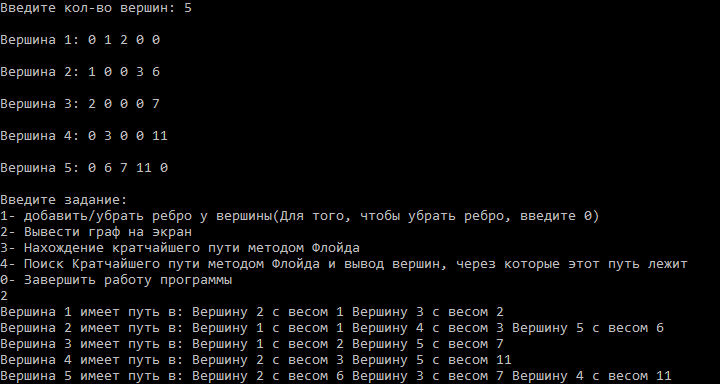


Рис. 5 - Создание и вывод графа первого теста

А теперь с помощью функции добавления ребер текущий граф сводится в графу №1 (Рис. 6)

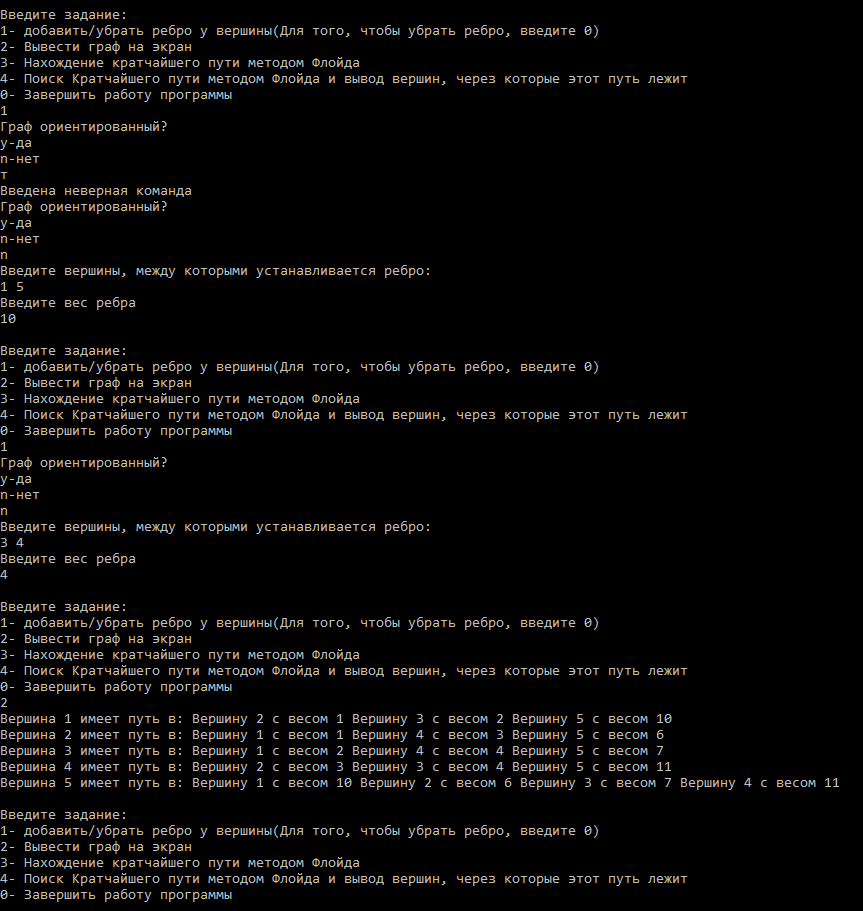


Рис. 6 - Приведение графа первого теста к графу №1 с помощью добавления рёбер и вывод на экран

Соответственно сам граф №1 (Рис. 7)

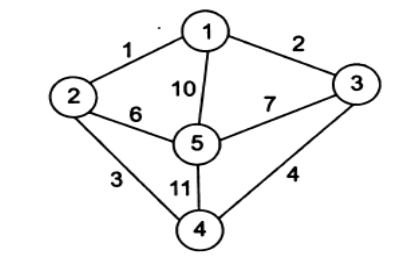


Рис. 7 - Граф №1

Далее с помощью метода Флойда находим кратчайшие пути к различным вершинам (Рис. 8).

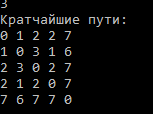


Рис. 8 – Кратчайшие пути от различных вершин

И теперь находим кратчайшее расстояние между вершинами, а также вершины, что находятся на пути, сначала введём вершины 3 и 2 (Рис. 9)

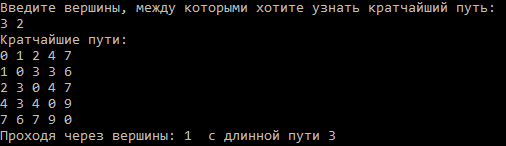


Рис. 9 – Поиск кратчайшего пути между вершинами 3 и 2

Также найдем путь кратчайший путь между соседними вершинами 1 и 3 (Рис. 10).

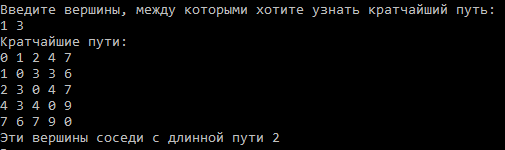


Рис. 10 - Поиск кратчайшего пути между вершинами 1 и 3

Второй тест проведём на ориентированном графе №5, удалив из него пару рёбер, что приведёт его к новому виду (Рис. 11).

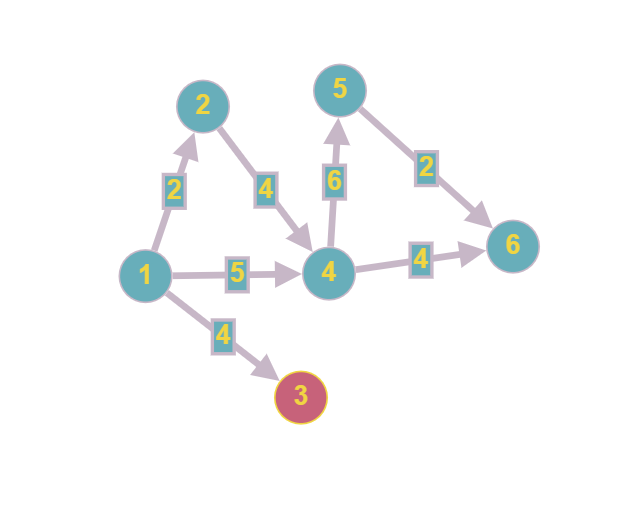


Рис. 11 - Граф второго теста

Для этого введём в программу первоначально координаты этого графа и выведем его на экран (Рис. 12).

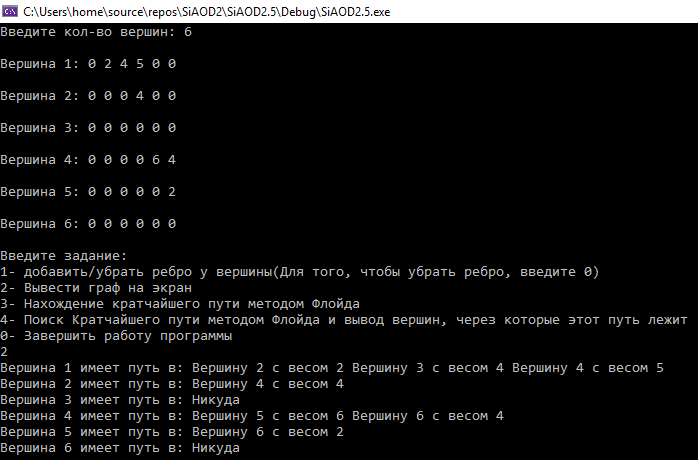


Рис. 12 - Создание и вывод графа первого теста

А теперь с помощью функции добавления ребер текущий граф сводится в графу №5 (Рис. 13)

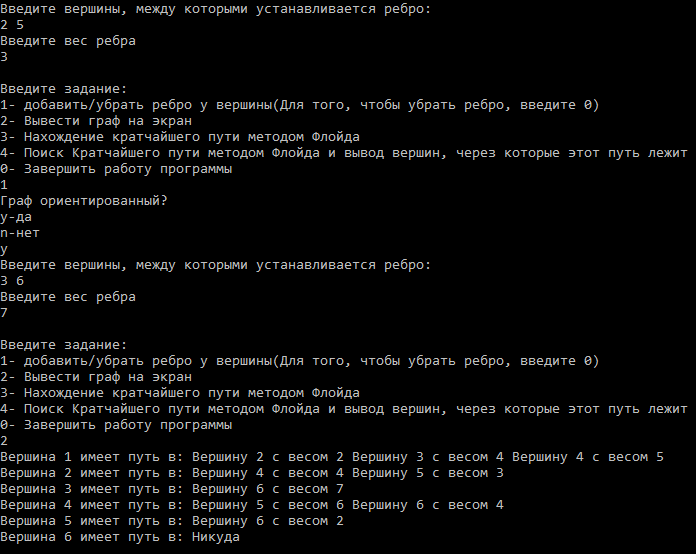


Рис. 13 - Приведение графа второго теста к графу №5 с помощью добавления рёбер и вывод на экран

Соответственно сам граф №5 (Рис. 14)

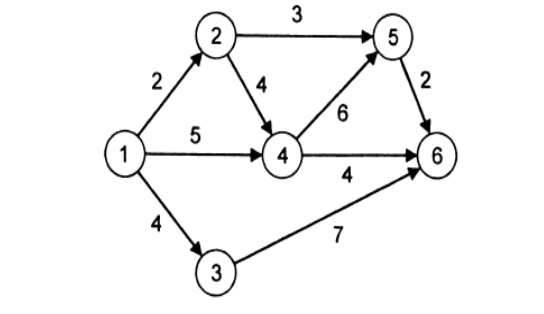


Рис. 14 - Граф №5

Далее с помощью метода Флойда находим кратчайшие пути к различным вершинам (Рис. 15).

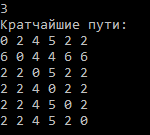


Рис. 15 – Кратчайшие пути от различных вершин

И теперь находим кратчайшее расстояние между вершинами, а также вершины, что находятся на пути, сначала введём вершины 1 и 6 (Рис. 16)

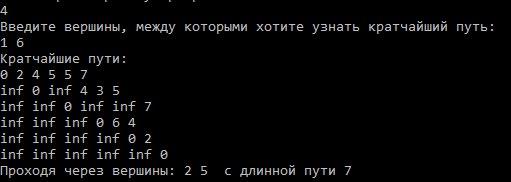


Рис. 16 – Поиск кратчайшего пути между вершинами 1 и 6

Также найдем путь кратчайший путь между соседними вершинами 6 и 1 и увидим, что программа выводит сообщение о несуществовании этого пути (Рис. 17).

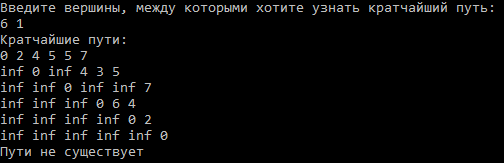


Рис. 17 - Поиск кратчайшего пути между вершинами 6 и 1

Из результатов выполнения программы видно:

Тестирование графа проведено успешно. При вводе любого графа программа работает корректно.

# **Вывод**

В результате выполнения работы я получила практические навыки по выполнению операций над структурой данных граф.

# **Исходный код программы**

|  |
| --- |
| #include <iostream>  using namespace std;  const int maxint = 2147483;  class Graf  {  public:  struct Node  {  struct List  {  List\* next = NULL;  int weight = 0;  int number = 0;  };  List\* Head = NULL;  List\* Last = NULL;  void AddNew(int weight, int number)  {  List\* list = new List;  list->weight = weight;  list->number = number;  if (!Head)  {  Head = list;  Last = list;  }  else  {  Last->next = list;  Last = list;  }  }  void ChangeValue(int to, int p)  {  List\* list = Head;  for (int i = 0; i != to; i++)  {  if (list->next)  list = list->next;  else  {  cout << "Введена неверная конечная вершина";  return;  }  }  list->weight = p;  }  void PrintList()  {  int c = 0;  List\* list;  for (list = Head; list->next; list = list->next)  {  if (list->weight != 0)  {  cout << "Вершину " << list->number << " с весом " << list->weight << " ";  c++;  }  }  if (list->weight != 0)  {  cout << "Вершину " << list->number << " с весом " << list->weight;  c++;  }  if (c == 0)  cout << "Никуда";  }  void DeleteList()  {  List\* list = Head;  while (true)  {  if (list->next)  {  list = list->next;  delete Head;  Head = list;  }  else  {  delete Head;  break;  }  }  }  };  Graf()  {  int weight;  cout << "Введите кол-во вершин: ";  cin >> n;  matr = new Node[n];  for (int i = 0; i < n; i++)  {  cout << "\nВершина " << i+1 << ": ";  for (int j = 0; j < n; j++)  {  cin >> weight;  matr[i].AddNew(weight, j+1);  }  }  }  void AddEdge(int from, int to, int p, bool f)  {  if (f)  {  matr[from-1].ChangeValue(to-1, p);  }  else  {  matr[from - 1].ChangeValue(to - 1, p);  matr[to - 1].ChangeValue(from - 1, p);  }  }  void GetGraf()  {  for (int i = 0; i < n; i++)  {  cout << "Вершина " << i + 1 << " имеет путь в: ";  matr[i].PrintList();  cout << "\n";  }  }  void NameH(int\*\* b, int from, int to)  {  int k;  k = b[from][to];  if (k == 0)  return;  NameH(b, from, k);  NameH(b, k, to);  }  void Floyd(bool f, int from, int to)  {  Node::List\* l;  int i, j;  int\*\* a = new int\* [n+1];  if (f)  {  int\*\* b = new int\* [n+1];  for (i = 1; i <= n; i++)  {  a[i] = new int[n+1];  b[i] = new int[n+1];  }  for (i = 1; i <= n; i++)  for (j = 1; j <= n; j++)  {  a[i][j] = 0;  b[i][j] = 0;  }  for (i = 1; i <= n; i++)  {  l = matr[i-1].Head;  j = 1;  while (true)  {  if ((i != j) && (l->weight == 0))  a[i][j] = maxint;  else  a[i][j] = l->weight;  if (l->next)  l = l->next;  else  break;  j++;  }  }  for (int k = 1; k <= n; k++)  for (i = 1; i <= n; i++)  for (j = 1; j <= n; j++)  {  if (a[i][k] + a[k][j] < a[i][j])  {  a[i][j] = a[i][k] + a[k][j];  b[i][j] = k;  }  }  cout << "Кратчайшие пути:\n";  for (i = 1; i <= n; i++)  {  for (j = 1; j <= n; j++)  {  if (a[i][j] == maxint)  cout << "inf" << " ";  else  cout << a[i][j] << " ";  }  cout << "\n";  }  if (a[from][to] != maxint)  {  if (b[from][to] != 0)  {  cout << "Проходя через вершины: ";  NameH(b, from, to);  delete[] b;  }  else  cout << "Эти вершины соседи";  cout << " c длинной пути " << a[from][to];  }  else  cout << "Пути не существует";  }  else  {  for (i = 0; i < n; i++)  a[i] = new int[n];  for (i = 0; i < n; i++)  for (j = 0; j < n; j++)  a[i][j] = 0;  for (i = 0; i < n; i++)  {  l = matr[i].Head;  j = 0;  while (true)  {  a[i][j] = l->weight;  if (l->next)  l = l->next;  else  break;  j++;  }  }  for (int k = 0; k < n; k++)  for (i = 0; i < n; i++)  for (j = 0; j < n; j++)  if ((a[i][k] + a[k][j] < a[i][j]) || ((i != j) && (a[i][j] == 0)))  a[i][j] = a[i][k] + a[k][j];  cout << "Кратчайшие пути:\n";  for (i = 0; i < n; i++)  {  for (j = 0; j < n; j++)  cout << a[i][j] << " ";  cout << "\n";  }  }  delete[] a;  }  ~Graf()  {  delete[] matr;  }  private:  int n;  Node\* matr;  };  int main()  {  setlocale(LC\_ALL, "RUSSIAN");  Graf g;  int number;  start:  cout << "\nВведите задание: \n1- добавить/убрать ребро у вершины(Для того, чтобы убрать ребро, введите 0) \n2- Вывести граф на экран\n3- Нахождение кратчайшего пути методом Флойда\n4- Поиск Кратчайшего пути методом Флойда и вывод вершин, через которые этот путь лежит\n0- Завершить работу программы\n";  cin >> number;  switch (number)  {  case 1:  {  int from, to, p;  char c;  bool f = false;  while(true)  {  cout << "Граф ориентированный?\ny-да\nn-нет\n";  cin >> c;  if (c == 'y' || c == 'Y')  {  f = true;  break;  }  if (c == 'n' || c == 'N')  {  f = false;  break;  }  cout << "Введена неверная команда\n";  }  cout << "Введите вершины, между которыми устанавливается ребро:\n";  cin >> from >> to;  cout << "Введите вес ребра\n";  cin >> p;  g.AddEdge(from, to, p, f);  goto start;  }  case 2:  {  g.GetGraf();  goto start;  }  case 3:  {  g.Floyd(false, 0, 0);  goto start;  }  case 4:  {  int from, to;  cout << "Введите вершины, между которыми хотите узнать кратчайший путь:\n";  while (true) {  cin >> from >> to;  if (from != to)  break;  else  cout << "Пути точки самой в себя не существует, введите значения ещё раз:\n";  }  g.Floyd(true, from, to);  goto start;  }  default:  break;  }  } |