Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования

# Построение цепочки переводчиков

## ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

по дисциплине «Алгоритмы и структуры данных»
ЮУрГУ–010302.2020.152.ПЗ КР

Автор работы,
студент группы ЕТ-212
_____ / Д.П. Чумаков
«_26_» _мая_____ 2020 г.

Руководитель работы,
старший преподаватель
_____ / А.С. Шелудько
«____» _____ 2020 г.

Работа защищена с оценкой
_____
«____» _____ 2020 г.

Челябинск 2020

# АННОТАЦИЯ

Чумаков Д.П. Построение цепочки переводчиков. – Челябинск:ЮУрГУ, ЕТ-212, 2020. – 31 с., 7 ил., библиогр. список – 6 наим., 3 прил.

Целью работы является разработка консольного приложения для построения цепочки переводчиков с использованием шаблонных алгоритмов и структур данных на языке программирования C++.

В первом разделе приводится постановка задачи о построении цепочки переводчиков. Во втором разделе описывается алгоритм решения данной задачи и обосновывается выбор шаблонных алгоритмов и структур данных. Третий раздел включает в себя описание программы, решающей данную задачу. В приложениях приведены исходный код программы, руководство пользователя, пример работы программы.

ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

**Цель работы**: разработать консольное приложение для построения цепочки переводчиков на языке программирования C++.

**Задачи работы**:

– привести описание графа, который визуализирует данную задачу;

– разработать модуль с функцией обхода графа в ширину;

– разработать модуль с реализацией структуры данных очередь, которая будет поддерживать следующие методы: вставка в конец очереди, просмотр и удаление первого элемента очереди;

– разработать модуль с реализацией структуры данных массив, который будет поддерживать следующие методы: вставка элемента в конец массива, обращение по индексу, запрос количества элементов в массиве;

– оценить порядок быстродействия как структур данных, так и алгоритма решения задачи.

# 1 ПОСТАНОВКА ЗАДАЧИ

Задано множество из N людей и множество из M языков. Для любой пары людей A и B найти кратчайшую цепочку переводчиков, которая обеспечит передачу сообщения от A к B. Пример цепочки переводчиков изображен на рисунке 1.1.

Исходные данные в программе считываются из текстового файла input.txt. Данные в файле записаны следующим образом: В первой строке находится единственное число N – количество людей. Далее в последующих N строках находятся непустые строки – имена людей, то есть в i-ой строке содержится имя i-го человека ($i = \overline{1, N}$). В следующей строке находится единственное число M – количество языков. В последующих M строках находятся непустые строки – названия языков, то есть в j-ой строке содержится название j-го языка ($j = \overline{1, M}$).

Далее следует список смежности, представленный в виде N строк, где каждая i-ая строка (i-ый человек) содержит $c_i$ чисел $p_k$ (знает некоторое количество языков) и «0» в конце, как признак окончания строки. Здесь $i = \overline{1, N}$, $k = \overline{1, c_i}$, $p_k \in \{1, \dots, M\}$.

Пример содержимого входного файла:

```
4
Петя
Вася
Иван
Сережа
3
русский
болгарский
английский
1 0
1 2 0
2 3 0
3 0
```

В данном примере Петя знает только русский язык, Вася знает русский и болгарский, Иван – болгарский и английский, Сережа – только английский.

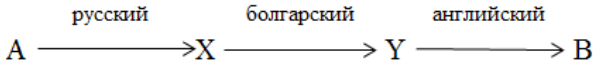


Рисунок 1.1 – Цепочка переводчиков

## 2 АЛГОРИТМ РЕШЕНИЯ

Считаем, что каждый из людей владеет не менее чем одним языком, а для каждого языка найдется хотя бы один его носитель. В данной задаче можно выделить двудольный граф, первой долей которого являются люди, а второй долей – языки. Ребро соединяет соответствующего человека и язык, если человек знает его. Для удобства введем общую нумерацию вершин графа, то есть вершины первой доли (люди) нумеруются от 1 до N, а вершины второй доли (языки) – от N+1 до N+M.

На рисунке 2.1 показан граф, соответствующий примеру из постановки задачи. Соответственно вершина 1 – Петя, 2 – Вася, 3 – Иван, 4 – Сережа, 5 – русский язык, 6 – болгарский язык, 7 – английский язык.

Тогда решение данной задачи сводится к нахождению кратчайшего пути от вершины-человека A до вершины-человека B.

Существуют алгоритмы по нахождению кратчайшего пути в взвешенном графе, время работы которых порядка $O((N + M)^2)$. Граф в этой задаче невзвешенный, поэтому кратчайший путь можно найти обходом графа в ширину, предварительно заведя массив предков и по нему восстановив путь. Порядок времени работы данного решения составляет $O(N + M + E)$, где E – количество ребер в графе, что значительно лучше, чем порядок времени работы алгоритмов непосредственного поиска кратчайшего пути. На рисунке 2.2 представлена блок-схема алгоритма решения.

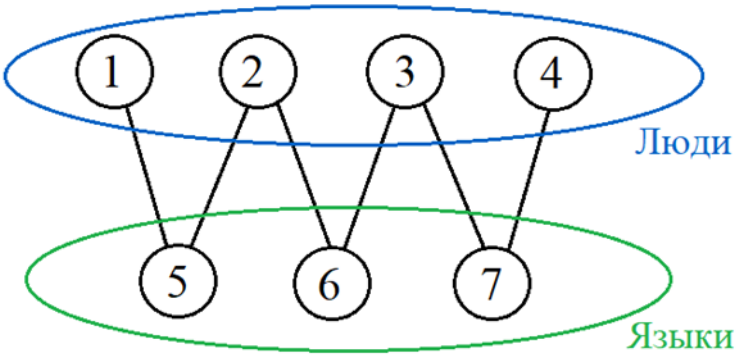


Рисунок 2.1 – Граф


Рисунок 2.2 – Блок-схема решения

В процессе обхода графа в ширину потребуется находить всех соседей для очередной вершины, поэтому будем хранить граф как список смежности, так как данная операция при таком способе хранения будет выполняться быстрее по сравнению с матрицей смежности. К тому же список смежности использует только необходимое количество памяти.

Для хранения графа списком смежности воспользуемся двумерным массивом G чисел, количество строк которого N+M, причем количество элементов в i-ой строке зависит от того, сколько соседей у вершины i. На рисунке 2.3 представлена схема такого массива на примере графа, изображенного на рисунке 2.1.

Если массив будет представлен как последовательность, то порядок времени выполнения операций:

– произвольного доступа по индексу порядка $O(1)$;

– вставки элемента в конец массива – среднее время порядка $O(1)$, худшее время порядка $O(n)$, где n – количество элементов в массиве.

Конечно, для улучшения времени вставки в конец можно воспользоваться линейным списком или какими-либо другими структурами данных, однако они только ухудшат время произвольного доступа по индексу в массиве. К тому же, вставка в конец осуществляется только перед обработкой основных запросов, а произвольный доступ нужен при каждом запросе. Поэтому лучше реализовать массив как последовательность.

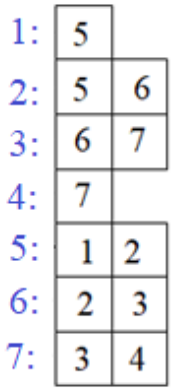


Рисунок 2.3 – Список смежности

При обходе графа в ширину используется такая структура данных как очередь. Для реализации очереди можно использовать однонаправленный список с головой, которая будет указывать либо на несуществующий элемент, либо на первый элемент очереди; также будет использоваться дополнительный указатель на последний элемент. Все операции с очередью (вставка в конец очереди, просмотр и удаление первого элемента очереди) при данной реализации осуществляются за время порядка $O(1)$. Кроме того, для хранения очереди используется только необходимое количество памяти. На рисунке 2.4 изображена очередь из 3 элементов.

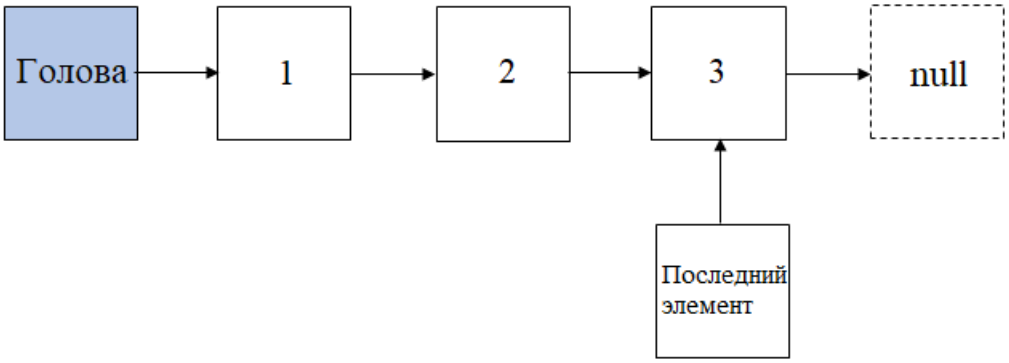


Рисунок 2.4 – Очередь

# 3 ОПИСАНИЕ ПРОГРАММЫ

Файл Vector.h

В ходе создания программы был разработан шаблонный класс Vector<T>, упрощенный аналог класса std::vector<T> из STL. Ниже представлен интерфейс данного класса. Реализация методов представлена в приложении 1.

```
template <typename T> // подобие std::vector<>
class Vector
{
	// начальное количество элементов по умолчанию
	static const int N_MIN = 4;

	// текущее количество элементов
	int n;

	// текущее количество единиц зарезервированной памяти
	int nmax;

	// указатель на область памяти
	// зарезервированной памяти для вектора
	T* a;
public:
	// количество элементов и аргумент для инициализации
	Vector(int, T);

	Vector(int);   // количество элементов
	Vector();      // конструктор по умолчанию

	// конструктор копий
	Vector(const Vector<T>&);

	// запрет присваивания
	Vector<T>& operator=(const Vector<T>&) = delete;

	T& operator[](int);
	T operator[](int) const;
	void push_back(T); // вставка в конец
	void reserve(int); // резервирование памяти
	int size() const { return n; };
	~Vector() { delete[] a; }
};
```

Ниже приведен фрагмент программы, в котором объявляется граф.

```
Vector<Vector<int> > graph(n_vertex);
for (int i = 0; i < n; i++) {
     int a;
     fin >> a;
     while (a) {
          graph[i].push_back(a - 1 + n);
          graph[a - 1 + n].push_back(i);
          fin >> a;
     }
}
```

В данном фрагменте кода n_vertex – количество вершин в графе, равное сумме количества людей и языков, fin – объект, считывающий список смежности из входного файла, n – количество людей.

Файл Queue.h

Также был разработан шаблонный класс Queue<T>, аналог класса std::queue<T> из STL. Ниже представлен интерфейс данного класса. Реализация методов представлена в приложении 1.

```
template <typename T>
struct Node // узел списка
{
     T data;         // данные
     Node<T>* next; // указатель на следующий элемент
};

template <typename T> // подобие std::queue<>
class Queue
{
     Node<T>* head; // указатель на голову списка
     Node<T>* last; // указатель на последний
                    // элемент списка
public:
     Queue();

     // запрет на копирование
     Queue(const Queue<T>&) = delete;

     // запрет на присваивание
     Queue<T>& operator=(const Queue<T>&) = delete;

     void push(T);
     bool is_empty() const { return head == last; };
     T front() const;
     void pop();
     ~Queue();
};
```

Ниже приведен фрагмент кода, в котором используется этот класс.

```
Queue<int> q;
q.push(a);
while (!q.is_empty()) {
     int v = q.front();
     q.pop();
     //…
}
```

Файл String.h

Для удобства работы со строками был разработан класс String, упрощенный аналог класса std::string из STL. Ниже приведен интерфейс этого класса.

```
class String // подобие std::string
{
     int len;       // длина строки
     char* str;     // указатель на область памяти
public:
     String(const char* = "");
     String(const String&);
     String(char);
     ~String() { delete[] str; };

     int length() const { return len; }
     const char* c_str() const { return str; }

     String& operator = (const String&);
     char operator [] (int) const;
     char& operator [] (int);

     // ввод до конца строки
     friend istream& operator >> (istream&, String&);
     friend ostream& operator << (ostream&, const String&);

     String& operator += (const String&);
};
```

Файл Translaters.h

Функция main и функция BFS (функция обхода графа в ширину) размещены в файле Translaters.cpp (см. приложение 1). Ниже приведено описание функции BFS.

```
int BFS(int a, int b, const Vector<Vector<int> >& g,
          Vector<int>& chain);
```

Здесь a, b – номера первого и последнего человека соответственно, между которыми нужно построить цепочку переводчиков, g – сам граф, chain

– массив, в котором будут храниться номера вершин, образующих цепочку переводчиков. Функция BFS возвращает количество элементов массива chain, если цепочка между a и b существует, иначе возвращает 0.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были выбраны оптимальные структуры данных и успешно внедрены в алгоритм решения. После анализа задачи структуры данных были реализованы на языке C++. Разработанный код был проверен на контрольных тестах и в код были внесены необходимые исправления. Для консольного приложения была разработана документация, описывающая ее установку и использование. Таким образом, цель работы была достигнута, задачи – решены.

Результаты работы можно использовать в процессе последующего обучения в форме навыков практического применения алгоритмов и структур данных для разработки сложных программных систем, понимания порядка этапов разработки программного обеспечения и достигаемых на каждом этапе результатов.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ахо, А.В. Структуры данных и алгоритмы / А.В. Ахо, Д.Э. Хопкрофт, Д.Д. Ульман. – М.: Вильямс, 2000. – 382 с.

2. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт. – М.: Мир, 1989. – 360 с.

3. Кормен, Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – М.: МЦНМО, 2001. – 955 с.

4. Кнут, Д. Искусство программирования. Том 3. Сортировка и поиск / Д. Кнут. – М.: Вильямс, 2000. – 822 с.

5. Подбельский, В.В. Курс программирования на языке Си / В.В. Подбельский, С.С. Фомин. – М.: ДМК Пресс, 2012. – 384 с.

6. Хаггарти, Р. Дискретная математика для программистов / Р. Хаггарти. – М.: Техносфера, 2012. – 399 с.

# ПРИЛОЖЕНИЕ 1

## Текст программы

### Файл Translaters.cpp

```
#define _CRT_SECURE_NO_WARNINGS

#include <iostream>
#include <fstream>
#include <windows.h>

#include "Vector.h"
#include "String.h"
#include "Queue.h"

using namespace std;

int BFS(int a, int b, const Vector<Vector<int> >& g,
          Vector<int>& chain)
{
     // a, b - номера первого и
     // последнего человека соответственно
     // между которыми нужно построить цепочку переводчиков

     // g - сам граф

     // chain - массив, в котором будут храниться
     // номера вершин, образующих цепочку переводчиков

     // BFS возвращает количество элементов массива chain,
     // если цепочка между a и b существует,
     // иначе возвращает 0
     int n = g.size(); // количество вершин в графе
     Vector<bool> used(n, false);
     Vector<int> p(n, -1);
     used[a] = true;
     Queue<int> q;
     q.push(a);
     int v = a;
     while (!q.is_empty()) {
          v = q.front();
          q.pop();
          if (v == b) {
               break;
          }
```

```
            for (int i = 0; i < g[v].size(); i++) {
                int u = g[v][i];
                if (!used[u]) {
                    used[u] = true;
                    p[u] = v;
                    q.push(u);
                }
            }
        }
        if (v == b) {
            int i = 0;
            Vector<int> chain_1(n);
            for (int u = b; u != -1; u = p[u]) {
                chain_1[i++] = u;
            }
            for (int j = i - 1; j >= 0; j--) {
                chain[i - 1 - j] = chain_1[j];
            }
            return i;
        }
        return 0;
}

int main()
{
        SetConsoleOutputCP(1251);

        ifstream fin("input.txt");
        if (!fin.is_open()) {
            cout << "Ошибка открытия входного файла\n";
            cin.get();
            return 0;
        }

        int n, m; // количество людей, языков соответственно
        fin >> n;
        Vector<String> names(n);
        cout << "Имена:\n";
        for (int i = 0; i < n; i++) {
            fin >> names[i];
            cout << i + 1 << ". " << names[i] << "\n";
        }

        fin >> m;
        Vector<String> languages(m);
        cout << "\nЯзыки:\n";
        for (int i = 0; i < m; i++) {
            fin >> languages[i];
            cout << i + 1 << ". " << languages[i] << "\n";
        }
        cout << "\n";

        int n_vertex = n + m;
```

```
// общая нумерация в графе,
// представимого в виде списка смежности:
// 0...(n_people - 1) - люди
// n_people...(n_vertex - 1) - языки
Vector<Vector<int> > graph(n_vertex);
// для уменьшения количества
// перераспределения памяти
for (int i = 0; i < n_vertex; i++) {
     graph[i].reserve(100);
}
for (int i = 0; i < n; i++) {
     int a;
     fin >> a;
     while (a) {
          graph[i].push_back(a - 1 + n);
          graph[a - 1 + n].push_back(i);
          fin >> a;
     }
}
fin.close();

Vector<int> chain(n_vertex);
// массив, в котором будут храниться номера вершин,
// образующих цепочку переводчиков
cout << "Введите номер пункта меню\n";
while (true) {
     cout << "1. Запрос\n2. Завершить\n";
     char c;
     cin >> c;
     if (c == '2') {
          // условие окончания работы
          break;
     }
     if (c != '1') {
          continue;
     }
     cout << "\nВведите номера двух людей через пробел\n";
     int a, b; // номера людей, между которыми нужно
                    // построить цепочку переводчиков
     cin >> a >> b;
     if (a == b || a > n || b > n ||
          a <= 0 || b <= 0) {
          cout << "Некорректный запрос\n\n";
          continue;
     }
     int n_chain = BFS(a - 1, b - 1, graph, chain);
     for (int i = 0; i < n_chain; i++) {
          if (chain[i] < n) {
               // если номер вершины означает человека
               cout << names[chain[i]];
          }
```

```
                else {
                    // иначе номер вершины означает язык
                    cout << languages[chain[i] - n];
                }
                if (i < n_chain - 1) {
                    cout << "->";
                }
            }
            if (n_chain == 0) {
                cout << "Цепочки переводчиков нет";
            }
            cout << "\n\nДля продолжения нажмите клавишу Enter\n";
            cin.get();
            cin.get();
        }
        return 0;
}
```

Файл Vector.h

```
#ifndef VECTOR_H
#define VECTOR_H

class BadIndex {};

template <typename T> // подобие std::vector<>
class Vector
{
     // начальное количество элементов по умолчанию
     static const int N_MIN = 4;

     // текущее количество элементов
     int n;

     // текущее количество единиц зарезервированной памяти
     int nmax;

     // указатель на область памяти
     // зарезервированной памяти для вектора
     T* a;
public:
     // количество элементов и аргумент для инициализации
     Vector(int, T);

     Vector(int);   // количество элементов
     Vector();      // конструктор по умолчанию

     // конструктор копий
     Vector(const Vector<T>&);

     // запрет присваивания
     Vector<T>& operator=(const Vector<T>&) = delete;

     T& operator[](int);
     T operator[](int) const;
     void push_back(T); // вставка в конец
     void reserve(int); // резервирование памяти
     int size() const { return n; };
     ~Vector() { delete[] a; }
};

template <typename T>
Vector<T>::Vector(int n, T value)
{
     this->n = n;
     nmax = n;
     a = new T[nmax];
     for (int i = 0; i < n; i++) {
          a[i] = value;
     }
}
```

```
template <typename T>
Vector<T>::Vector(int n)
{
     this->n = n;
     nmax = n;
     a = new T[nmax];
}
template <typename T>
Vector<T>::Vector()
{
     n = 0;
     nmax = N_MIN;
     a = new T[nmax];
}

template <typename T>
Vector<T>::Vector(const Vector<T>& v)
{
     n = v.n;
     nmax = v.nmax;
     a = new T[nmax];
     for (int i = 0; i < n; i++) {
          a[i] = v.a[i];
     }
}

template <typename T>
T& Vector<T>::operator[](int i)
{
     if (i < 0 || i >= n) {
          throw BadIndex();
     }
     return a[i];
}

template <typename T>
T Vector<T>::operator[](int i) const
{
     if (i < 0 || i >= n) {
          throw BadIndex();
     }
     return a[i];
}
```

```
template <typename T>
void Vector<T>::push_back(T value)
{
     if (n == nmax) {
          nmax *= 2;
          T* b = new T[nmax];
          for (int i = 0; i < n; i++) {
               b[i] = a[i];
          }
          delete[] a;
          a = b;
     }
     a[n++] = value;
}

template <typename T>
void Vector<T>::reserve(int _nmax)
{
     if (_nmax <= nmax) {
          return;
     }
     nmax = _nmax;
     T* b = new T[nmax];
     for (int i = 0; i < n; i++) {
          b[i] = a[i];
     }
     delete[] a;
     a = b;
}
#endif // VECTOR_H
```

Файл Queue.h

```
#ifndef QUEUE_H
#define QUEUE_H
class BadEmpty {};
template <typename T>
struct Node // узел списка
{
     T data;        // данные
     Node<T>* next; // указатель на следующий элемент
};

template <typename T> // подобие std::queue<>
class Queue
{
     Node<T>* head; // указатель на голову списка
     Node<T>* last; // указатель на последний
                    // элемент списка
public:
     Queue();

     // запрет на копирование
     Queue(const Queue<T>&) = delete;

     // запрет на присваивание
     Queue<T>& operator=(const Queue<T>&) = delete;

     void push(T);
     bool is_empty() const { return head == last; };
     T front() const;
     void pop();
     ~Queue();
};

template <typename T>
Queue<T>::Queue()
{
     head = new Node<T>;
     head->next = nullptr;
     last = new Node<T>;
     last = head;
}

template <typename T>
void Queue<T>::push(T data)
{
     Node<T>* x = new Node<T>;
     x->data = data;
     x->next = nullptr;
     last->next = x;
     last = x;
}
```

```
template <typename T>
T Queue<T>::front() const
{
     if (is_empty()) {
          throw BadEmpty();
     }

     return head->next->data;
}

template <typename T>
void Queue<T>::pop()
{
     if (is_empty()) {
          throw BadEmpty();
     }
     Node<T>* p = head->next;
     if (p == last) {
          last = head;
     }
     head->next = p->next;
     delete p;
}

template <typename T>
Queue<T>::~Queue()
{
     Node<T>* p = head, * x;
     while (p != nullptr) {
          x = p;
          p = p->next;
          delete x;
     }
}
#endif // QUEUE_H
```

Файл String.h

```
#ifndef STRING_H
#define STRING_H

#include <iostream>

using namespace std;

class String // подобие std::string
{
     int len;       // длина строки
     char* str;     // указатель на область памяти
public:
     String(const char* = "");
     String(const String&);
     String(char);
     ~String() { delete[] str; };

     int length() const { return len; }
     const char* c_str() const { return str; }

     String& operator = (const String&);
     char operator [] (int) const;
     char& operator [] (int);

     // ввод до конца строки
     friend istream& operator >> (istream&, String&);
     friend ostream& operator << (ostream&, const String&);

     String& operator += (const String&);
};

#endif //STRING_H
```

Файл String.cpp

```
#define _CRT_SECURE_NO_WARNINGS

#include <cstring>

#include "String.h"

String::String(const char* s)
{
     len = strlen(s);
     str = new char[len + 1];
     strcpy(str, s);
}

String::String(const String& s)
{
     len = strlen(s.str);
     str = new char[len + 1];
     strcpy(str, s.str);
}

String::String(char c)
{
     len = 1;
     str = new char[2];
     str[0] = c;
     str[1] = '\0';
}

String& String::operator = (const String& s)
{
     String t(s);
     swap(len, t.len);
     swap(str, t.str);

     return *this;
}

char String::operator [] (int i) const
{
     if (i < 0 || i >= len)
     {
          cout << "Bad index\n";
          exit(1);
     }

     return str[i];
}
```

```
char& String::operator [] (int i)
{
     if (i < 0 || i >= len)
     {
          cout << "Bad index\n";
          exit(1);
     }

     return str[i];
}

istream& operator >> (istream& input, String& s)
{
     s = "";
     char c;
     do {
          input >> c;
          s += c;
     } while (input.peek() != '\n');

     return input;
}

ostream& operator << (ostream& output, const String& s)
{
     for (int i = 0; i < s.len; i++)
     {
          output << s[i];
     }

     return output;
}

String& String::operator += (const String& s)
{
     len += s.len;
     char* p = new char[len + 1];
     strcpy(p, str);
     strcat(p, s.str);

     delete str;
     str = p;
     return *this;
}
```

# ПРИЛОЖЕНИЕ 2

## Руководство пользователя

Порядок использования программы.

1. Создать в каталоге, где находится исполняемый файл Translaters.exe, текстовый файл с названием input.txt, если он не был создан раньше.

2. В текстовый файл внести данные в том формате, который описан в первом разделе пояснительной записки.

3. Закрыть файл, сохранив его.

4. Запустить исполняемый файл Translaters.exe.

5. На консоль выведутся нумерованные списки имен людей и названий языков. После чего предлагается выбрать один из двух пунктов меню («Запрос» и «Завершить») с помощью ввода номера пункта. Если выбран пункт «Завершить», программа завершит работу, иначе нужно действовать согласно пункту 6 руководства.

6. Для запроса введите номера двух людей через пробел, между которыми нужно найти цепочку переводчиков. После нажатия клавиши Enter, если цепочка существует, на экран будет выведена одна из кратчайших цепочек, иначе будет выведено сообщение «Цепочки переводчиков нет». После ознакомления с результатом нажмите еще раз Enter, и будет предложено снова выбрать один из двух пунктов меню (см. пункт 5 руководства).

# ПРИЛОЖЕНИЕ 3

## Результат выполнения программы

Содержимое файла input.txt:

```
6
Петя
Вася
Иван
Сережа
Максим
Женя
4
Русский
Английский
Немецкий
Болгарский
1 0
1 2 0
1 2 3 0
3 0
4 0
4 0
```

На рисунке П3.1 представлен граф, визуализирующий содержимое файла.

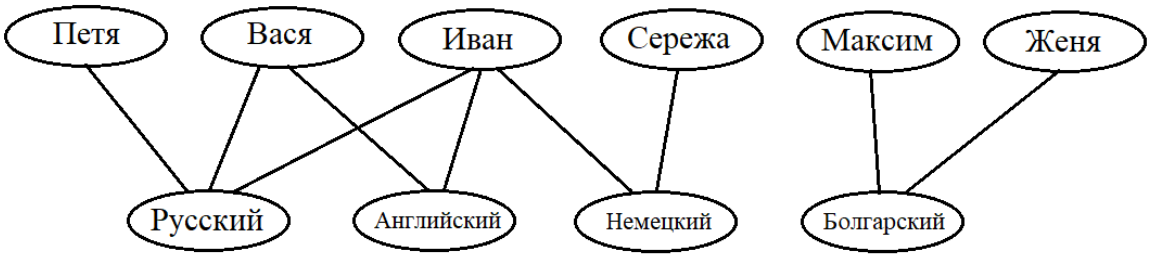


Рисунок П3.1 – Граф

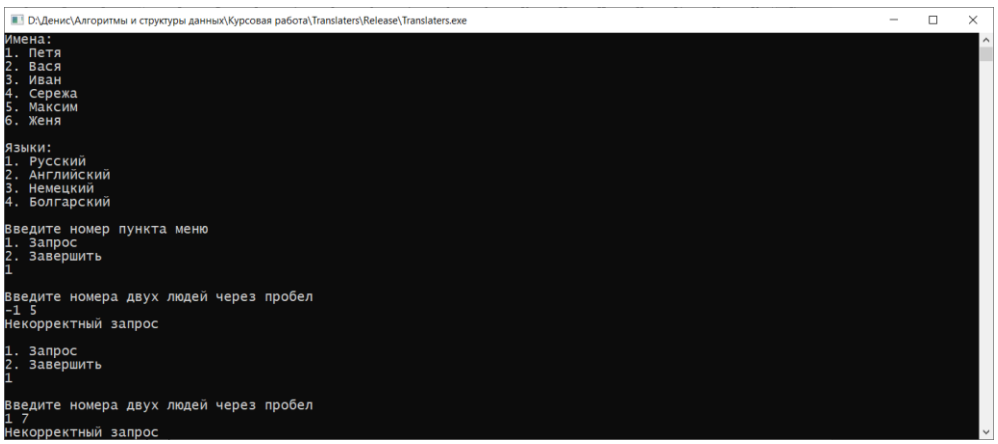


Рисунок П3.2 – Скриншот программы

Ниже представлен сокращенный текст результата работы программы (оставлены только сами запросы и ответы на них).

```
-1 5
Некорректный запрос

1 7
Некорректный запрос

1 1
Некорректный запрос

1 2
Петя->Русский->Вася

1 3
Петя->Русский->Иван

1 4
Петя->Русский->Иван->Немецкий->Сережа

1 5
Цепочки переводчиков нет

1 6
Цепочки переводчиков нет

2 1
Вася->Русский->Петя

2 3
Вася->Русский->Иван

2 4
Вася->Русский->Иван->Немецкий->Сережа

2 5
Цепочки переводчиков нет

2 6
Цепочки переводчиков нет

3 1
Иван->Русский->Петя

3 2
Иван->Русский->Вася

3 4
Иван->Немецкий->Сережа

3 5
Цепочки переводчиков нет
```

3 6
Цепочки переводчиков нет

4 1
Сережа->Немецкий->Иван->Русский->Петя

4 2
Сережа->Немецкий->Иван->Русский->Вася

4 3
Сережа->Немецкий->Иван

4 5
Цепочки переводчиков нет

4 6
Цепочки переводчиков нет

5 1
Цепочки переводчиков нет

5 2
Цепочки переводчиков нет

5 3
Цепочки переводчиков нет

5 4
Цепочки переводчиков нет

5 6
Максим->Болгарский->Женя

6 1
Цепочки переводчиков нет

6 2
Цепочки переводчиков нет

6 3
Цепочки переводчиков нет

6 4
Цепочки переводчиков нет

6 5
Женя->Болгарский->Максим