

Федеральное государственное бюджетное образовательное учреждение
высшего образования Нижегородский государственный технический
университет им. Р.Е. Алексеева

Лабораторная работа №3
«Алгоритм Дейкстры»

Выполнил:
студент 2-го курса
кафедры «Прикладная математика»
Поляков Кирилл

2020г.

Оглавление

Введение.....	3
Документация.....	4
Файл FSPG.cpp.....	4
Содержание FSPG.cpp.....	5
Файл Algorithms.h.....	6
Класс Algorithms.....	7
Содержание Algorithms.h.....	10
Файл List.h.....	12
Содержание List.h.....	13
Класс List.....	14
Содержание List.cpp.....	17
Файл Structs.h.....	18
Содержание Structs.h.....	19
Структура Way.....	20
Структура Node.....	22
Структура Data.....	24
Тестирование.....	26

Введение

Целью данной работы является - написать программу, реализующую алгоритм Дейкстры на произвольном графе, представленном в виде матрицы смежности или списка смежности.

Файл FSPG.cpp

```
#include <iostream>
#include "Algorithms.h"
```

Функции

```
int main (int argc, char *argv[])
```

Функции

◆ main()

```
int main ( int    argc,
           char * argv[]
        )
```

Входная точка программы

Аргументы

argc Количество аргументов командной строки

argv[] Аргументы командной строки

Возвращает

int

См. определение в файле **FSPG.cpp** строка **13**

FSPG.cpp

```
1  #include <iostream>
2
3  #include "Algorithms.h"
4
13 int main(int argc, char* argv[])
14 {
15     setlocale(LC_ALL, "rus");
16     Algorithms::FindWay(argc, argv);
17 }
```

Файл Algorithms.h

```
#include <string>
#include <iostream>
#include <list>
#include "List.h"
#include "Structs.h"
```

Классы

```
class Algorithms
```

Класс Algorithms

```
#include <Algorithms.h>
```

Открытые статические члены

```
static void FindWay (int argc, char *argv[])
```

Закрытые статические члены

```
static int ** ConvertToMatrix (list< string > edges, int sizeMatrix)
```

```
static List * ConvertToList (list< string > edges, int sizeMatrix)
```

```
static Data ReadFile (string filename)
```

```
template<class T >
```

```
static Way FindingWay (T matrix, const int sizeMatrix, int start, int finish)
```

Подробное описание

Класс, который представляет основной функционал.

См. определение в файле [Algorithms.h](#) строка **13**

Методы

◆ ConvertToList()

```
List * Algorithms::ConvertToList ( list< string > edges,  
                                int sizeMatrix  
                                )
```

static private

Функция, которая парсит входные данные и преобразует их в односвязанный список.

Аргументы

edges Строки, содержащие данные о ребрах и вершинах графа.

sizeMatrix Количество вершин графа.

Возвращает

List* представление графа в виде односвязного списка

См. определение в файле [Algorithms.cpp](#) строка **51**

◆ ConvertToMatrix()

```
int ** Algorithms::ConvertToMatrix ( list< string > edges,
                                     int          sizeMatrix
                                   )
```

static private

Функция, которая парсит входные данные и преобразует их в матрицу.

Аргументы

edges Строки, содержащие данные о ребрах и вершинах графа.
sizeMatrix Количество вершин графа.

Возвращает

int** представление графа в матричном виде

См. определение в файле [Algorithms.cpp](#) строка 7

◆ FindingWay()

```
template<class T >
```

```
Way Algorithms::FindingWay ( T          matrix,
                             const int  sizeMatrix,
                             int        start,
                             int        finish
                           )
```

static private

Функция, которая находит кратчайший путь от начальной вершины до конечной.

Аргументы

matrix Данные о вершинах.
sizeMatrix Количество вершин графа.
start Номер вершины, от которой будет построен путь.
finish Номер вершины, до которой будет построен путь.

Возвращает

Way

См. определение в файле [Algorithms.h](#) строка 63

◆ FindWay()


```
void Algorithms::FindWay ( int    argc,  
                          char * argv[]  
                          )
```

static

Основная функция, которая находит кратчайший путь от начальной вершины до конечной.

Аргументы

argc Количество элементов командной строки.

argv Аргументы командной строки.

См. определение в файле [Algorithms.cpp](#) строка **82**

◆ ReadFile()

```
Data Algorithms::ReadFile ( string filename )
```

static

private

Функция, которая считывает данные из файла и на их основании решает как хранить граф в памяти (matrix / link).

Аргументы

filename Имя файла, в котором хранятся данные.

Возвращает

Data

См. определение в файле [Algorithms.cpp](#) строка **138**

Объявления и описания членов классов находятся в файлах:

- [Algorithms.h](#)
- [Algorithms.cpp](#)

Algorithms.h

```
1  #pragma once
2  #include <string>
3  #include <iostream>
4  #include <list>
5  #include "List.h"
6  #include "Structs.h"
7
8  using namespace std;
9
13 static class Algorithms
14 {
15     static int** ConvertToMatrix(list<string> edges, int sizeMatrix);
16     static List* ConvertToList(list<string> edges, int sizeMatrix);
17
18     static Data ReadFile(string filename);
19
20     template <class T>
21     static Way FindingWay(T matrix, const int sizeMatrix, int start, int finish);
22
23 public:
24     static void FindWay(int argc, char* argv[]);
25 };
26
27 template <class T>
28 Way Algorithms::FindingWay(T matrix, const int sizeMatrix, int start, int finish)
29 {
30     Way way;
31
32     if (start <= 0 || start > sizeMatrix)
33     {
34         cout << "Error. The start point was entered incorrectly." << endl;
35         return way;
36     }
37
38     if (finish <= 0 || finish > sizeMatrix)
39     {
40         cout << "Error. The finish point was entered incorrectly." << endl;
41         return way;
42     }
43
44     int beg = start - 1;
45     int* distMin = new int[sizeMatrix];
46     int* vertex = new int[sizeMatrix];
47     int minIndex, min;
48
49     for (int i = 0; i < sizeMatrix; i++)
50     {
51         distMin[i] = INT_MAX;
52         vertex[i] = 1;
53     }
54     distMin[beg] = 0;
55
56     do
57     {
58         min = minIndex = INT_MAX;
59         for (int i = 0; i < sizeMatrix; i++)
60         {
61             if ((vertex[i] == 1) && (distMin[i] < min))
62             {
63                 min = distMin[i];
64                 minIndex = i;
65             }
66         }
67
68         if (minIndex != INT_MAX)
69         {
70             for (int i = 0; i < sizeMatrix; i++)
71             {
72                 if (matrix[minIndex][i] > 0)
73                 {
74                     int temp = min + matrix[minIndex][i];
75                     if (temp < distMin[i])
76                         distMin[i] = temp;
77                 }
78             }
79         }
80     } while (minIndex != finish);
81
82     return way;
83 }
```

```

112     }
113     }
114     vertex[minIndex] = 0;
115 }
116 } while (minIndex < INT_MAX);
117
118 int* ver = new int[sizeMatrix];
119 int end = finish - 1;
120 ver[0] = finish;
121 int ind = 1;
122 int weight = distMin[end];
123
124 way.length = distMin[end];
125
126 while (end != beg)
127 {
128     for (int i = 0; i < sizeMatrix; i++)
129         if (matrix[i][end] != 0)
130         {
131             int temp = weight - matrix[i][end];
132             if (temp == distMin[i])
133             {
134                 weight = temp;
135                 end = i;
136                 ver[ind++] = i + 1;
137             }
138         }
139     }
140
141 way.count = ind;
142 way.way = ver;
143 way.start = start;
144 way.finish = finish;
145
146 return way;
147 }
148

```

Файл List.h

```
#include "Structs.h"
```

Классы

```
class List
```

List.h

```
1 #pragma once
2 #include "Structs.h"
3
4
5
6
7 class List
8 {
9     Node* head = nullptr;
10    Node* last = nullptr;
11    int count = 0;
12 public:
13     void Add(int indexVertex, int weight);
14     void Print(int index);
15     int Count();
16
17
18
19     int& operator[] (const int i);
20 };
21
```

Класс List

```
#include <List.h>
```

Открытые члены

```
void Add (int indexVertex, int weight)
```

```
void Print (int index)
```

```
int Count ()
```

```
int & operator[] (const int i)
```

Закрытые данные

```
Node * head = nullptr
```

```
Node * last = nullptr
```

```
int count = 0
```

Подробное описание

Класс, реализующий односвязанный список.

См. определение в файле [List.h](#) строка 7

Методы

◆ Add()

```
void List::Add ( int indexVertex,  
                int weight  
                )
```

Функция, добавляющая новый элемент в список.

Аргументы

indexVertex Номер вершины.

weight Вес до вершины.

См. определение в файле [List.cpp](#) строка 7

◆ Count()

```
int List::Count ( )
```

Функция, которая возвращает количество элементов списка.

Возвращает

int Возвращает количество элементов списка.

См. определение в файле [List.cpp](#) строка 42

◆ operator[]()

```
int & List::operator[] ( const int i )
```

Перегруженный оператор [].

Аргументы

i Индекс выбранного элемента в списке.

Возвращает

int Возвращает выбранный элемент списка (0, если ничего не найдено).

См. определение в файле [List.cpp](#) строка 47

◆ Print()

```
void List::Print ( int index = -1 )
```

Функция, которая выводит значение списка.

Аргументы

index Номер вершины.

См. определение в файле [List.cpp](#) строка 27

Данные класса

◆ count

```
int List::count = 0
```

private

См. определение в файле [List.h](#) строка 11

◆ head

```
Node* List::head = nullptr
```

private

См. определение в файле [List.h](#) строка 9

◆ last

```
Node* List::last = nullptr
```

private

См. определение в файле [List.h](#) строка 10

Объявления и описания членов классов находятся в файлах:

- [List.h](#)
- [List.cpp](#)

List.cpp

```
1  #include "List.h"
2  #include <iostream>
3  #include <assert.h>
4
5  using namespace std;
6
7  void List::Add(int indexVertex, int weight)
8  {
9      count++;
10     if (head)
11     {
12         Node* temp = new Node();
13         temp->indexVertex = indexVertex;
14         temp->weight = weight;
15         last->next = temp;
16         last = temp;
17     }
18     else
19     {
20         head = new Node();
21         head->indexVertex = indexVertex;
22         head->weight = weight;
23         last = head;
24     }
25 }
26
27 void List::Print(int index = -1)
28 {
29     Node* temp = head;
30     if (index >= 0)
31         cout << "Vertex " << index + 1 << ": ";
32     while (temp)
33     {
34         cout << temp->indexVertex + 1;
35         if (temp->next)
36             cout << " -> ";
37         temp = temp->next;
38     }
39     cout << endl;
40 }
41
42 int List::Count()
43 {
44     return count;
45 }
46
47 int& List::operator[](const int i)
48 {
49     assert(i >= 0);
50     int notFind = 0;
51
52     if (i == 0)
53         return notFind;
54
55     Node* node = head;
56     while (node)
57     {
58         if (node->indexVertex == i)
59             return node->weight;
60         node = node->next;
61     }
62
63     return notFind;
64 }
65 }
```

Файл Structs.h

```
#include <list>
#include <string>
```

Классы

```
struct Way
```

```
struct Data
```

```
struct Node
```

Функции

```
void DeleteSpaces (string &str)
```

Функции

◆ DeleteSpaces()

```
void DeleteSpaces ( string & str )
```

Функция удаления первых пробелов из переданной строки.

Аргументы

str[out] строка, из которой будут удалены пробелы

См. определение в файле **Structs.h** строка **44**

Structs.h

```
1  #pragma once
2  #include <list>
3  #include <string>
4
5  using namespace std;
6
10 struct Way
11 {
12     int length = 0;
13     int* way = nullptr;
14     int count = 0;
15     int start = 0;
16     int finish = 0;
17 };
18
22 struct Data
23 {
24     int sizeGraph = 0;
25     list<string> values;
26     string typeConvert;
27 };
28
32 struct Node
33 {
34     int indexVertex;
35     int weight;
36     Node* next;
37 };
38
44 void DeleteSpaces(string& str)
45 {
46     size_t strBegin = str.find_first_not_of(' ');
47     size_t strEnd = str.find_last_not_of(' ');
48     str.erase(strEnd + 1, str.size() - strEnd);
49     str.erase(0, strBegin);
50 }
```

Структура Way

```
#include <Structs.h>
```

Открытые атрибуты

int **length** = 0

Длина пути [Подробнее...](#)

int * **way** = nullptr

Сам путь [Подробнее...](#)

int **count** = 0

Количесвто шагов [Подробнее...](#)

int **start** = 0

Вершина старта [Подробнее...](#)

int **finish** = 0

Вершина финиша [Подробнее...](#)

Подробное описание

Структура, которая хранит данные о найденном пути.

См. определение в файле **Structs.h** строка **10**

Данные класса

◆ count

```
int Way::count = 0
```

Количесвто шагов

См. определение в файле **Structs.h** строка **17**

◆ finish

```
int Way::finish = 0
```

Вершина финиша

См. определение в файле [Structs.h](#) строка 21

◆ length

```
int Way::length = 0
```

Длина пути

См. определение в файле [Structs.h](#) строка 13

◆ start

```
int Way::start = 0
```

Вершина старта

См. определение в файле [Structs.h](#) строка 19

◆ way

```
int* Way::way = nullptr
```

Сам путь

См. определение в файле [Structs.h](#) строка 15

Объявления и описания членов структуры находятся в файле:

- [Structs.h](#)

Структура Node

```
#include <Structs.h>
```

Открытые атрибуты

int **indexVertex**

Индекс вершины [Подробнее...](#)

int **weight**

Вес до вершины [Подробнее...](#)

Node * next

Указатель на следующую вершины [Подробнее...](#)

Подробное описание

Структура, которая хранит данные о вершинах графа.

См. определение в файле [Structs.h](#) строка 40

Данные класса

◆ indexVertex

int Node::indexVertex

Индекс вершины

См. определение в файле [Structs.h](#) строка 43

◆ next

Node* Node::next

Указатель на следующую вершины

См. определение в файле [Structs.h](#) строка 47

◆ weight

`int Node::weight`

Вес до вершины

См. определение в файле [Structs.h](#) строка **45**

Объявления и описания членов структуры находятся в файле:

- [Structs.h](#)

Структура Data

```
#include <Structs.h>
```

Открытые атрибуты

int **sizeGraph** = 0

Количество вершин графа [Подробнее...](#)

list< string > **values**

Данные о вершинах [Подробнее...](#)

string **typeConvert**

Тип хранения графа в памяти (matrix/list) [Подробнее...](#)

Подробное описание

Структура, которая хранит данные о графе.

См. определение в файле [Structs.h](#) строка 27

Данные класса

◆ sizeGraph

```
int Data::sizeGraph = 0
```

Количество вершин графа

См. определение в файле [Structs.h](#) строка 30

◆ typeConvert

```
string Data::typeConvert
```

Тип хранения графа в памяти (matrix/list)

См. определение в файле [Structs.h](#) строка 34

◆ values

`list<string> Data::values`

Данные о вершинах

См. определение в файле [Structs.h](#) строка 32

Объявления и описания членов структуры находятся в файле:

- [Structs.h](#)

Тестирование

```
0 7 6 0
0 0 4 5
0 0 0 9
0 0 0 0

Type of graph representation in memory: matrix.
Start: 1   Finish: 3
Way length: 6.
Count of steps: 2.
Way: 1 -> 3
```

```
Vertex 1: 2 -> 3
Vertex 2: 3 -> 4
Vertex 3: 4
Vertex 4:

Type of graph representation in memory: link.
Start: 1   Finish: 3
Way length: 6.
Count of steps: 2.
Way: 1 -> 3
```

```
4 link vertex_1 -> (2,7) -> (3, 6)
vertex_2 -> (3, 4) -> (4, 5)
vertex_3 -> (4, 9)
```