Белорусский государственный технологический университет

Кафедра Информационных Систем и Технологий

**Курс «Математическое программирование»**

**Лабораторная работа №5**

**АЛГОРИТМЫ НА ГРАФАХ**

**(алгоритмы поиска в ширину и глубину, топологическая сортировка)**

**Вариант 13**

Выполнила: Куницкий Н.Д.

ФИТ 2 курс 8 группа

Проверил: Бракович А.И.

Минск 2017

**Цель:** Освоить сущность и программную реализацию: а) способов представления графов; б) алгоритмов поиска в ширину и глубину; в) алгоритма топологической сортировки графов.

**Задание 1.**  Ориентированный граф **G** взять в соответствии с вариантом. Осуществить алгоритмы поиска в ширину и глубину, а также алгоритма топологической сортировки аналогично примерам, рассмотренным на лекциях.

13 вариант



**Алгоритм поиска в ширину:**

**Q-**очередь вершин (для промежуточного хранения вершин)

**С**- массив окраски вершин

**D**-массив расстояний

**P**-массив предшествующих вершин



|  |  |
| --- | --- |
| Q | 0 |

1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | G | W | W | W | W |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | 0 | I | I | I | I |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | N | N | N | N | N |

0

1

2

3

4

|  |  |
| --- | --- |
| Q | 1 |

2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | B | G | W | W | W |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | 0 | 1 | I | I | I |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | N | 0 | N | N | N |

0

1

2

3

4

|  |  |  |
| --- | --- | --- |
| Q | 2 | 3 |

3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | B | B | G | G | W |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | 0 | 1 | 2 | 2 | I |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | N | 0 | 1 | 1 | N |

0

1

2

3

4

|  |  |
| --- | --- |
| Q | 3 |

4.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | B | B | B | G | W |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | 0 | 1 | 2 | 2 | I |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | N | 0 | 1 | 1 | N |

0

1

2

3

4

|  |  |
| --- | --- |
| Q | 4 |

5.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | B | B | B | B | G |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | 0 | 1 | 2 | 2 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | N | 0 | 1 | 1 | 3 |

0

1

2

3

4

6.

|  |  |
| --- | --- |
| Q |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | B | B | B | B | B |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | 0 | 1 | 2 | 2 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | N | 0 | 1 | 1 | 3 |

0

1

2

3

4

BFS-дерево

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | N | 0 | 1 | 1 | 3 |

0

4

3

2

1

**Алгоритм поиска в глубину:**

**F-**время фиксации

**С**- массив окраски вершин

**D**-время обнаружения

**P**-массив предшествующих вершин

**t-**номер шага алгоритма



1.

|  |  |
| --- | --- |
| t | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | C | W | W | W | W |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | 1 | I | I | I | I |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | N | N | N | N | N |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| F | 0 | 0 | 0 | 0 | 0 |

0

1

2

3

4

2.

|  |  |
| --- | --- |
| t | 2 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | G | G | W | W | W |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | 1 | 2 | I | I | I |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | N | 0 | N | N | N |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| F | 0 | 0 | 0 | 0 | 0 |

0

1

2

3

4

3.

|  |  |
| --- | --- |
| t | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | G | G | G | W | W |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | 1 | 2 | 3 | I | I |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | N | 0 | 1 | N | N |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| F | 0 | 0 | 0 | 0 | 0 |

0

1

2

3

4

4.

|  |  |
| --- | --- |
| t | 4 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | G | G | G | G | W |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | 1 | 2 | 3 | 4 | I |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | N | 0 | 1 | 2 | N |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| F | 0 | 0 | 0 | 0 | 0 |

0

1

2

3

4

5.

|  |  |
| --- | --- |
| t | 5 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | G | G | G | G | G |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | 1 | 2 | 3 | 4 | 5 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | N | 0 | 1 | 2 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| F | 0 | 0 | 0 | 0 | 0 |

0

1

2

3

4

6.

|  |  |
| --- | --- |
| t | 6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | G | G | G | G | B |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | 1 | 2 | 3 | 4 | 5 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | N | 0 | 1 | 2 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| F | 0 | 0 | 0 | 0 | 6 |

0

1

2

3

4

7.

|  |  |
| --- | --- |
| t | 7 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | G | G | G | B | B |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | 1 | 2 | 3 | 4 | 5 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | N | 0 | 1 | 2 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| F | 0 | 0 | 0 | 5 | 6 |

0

1

2

3

4

8.

|  |  |
| --- | --- |
| t | 8 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | G | G | B | B | B |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | 1 | 2 | 3 | 4 | 5 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | N | 0 | 1 | 2 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| F | 0 | 0 | 8 | 5 | 6 |

0

1

2

3

4

9.

|  |  |
| --- | --- |
| t | 9 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | G | B | B | B | B |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | 1 | 2 | 3 | 4 | 5 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | N | 0 | 1 | 2 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| F | 0 | 9 | 8 | 5 | 6 |

0

1

2

3

4

10.

|  |  |
| --- | --- |
| t | 10 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | B | B | B | B | B |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | 1 | 2 | 3 | 4 | 5 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P | N | 0 | 1 | 2 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| F | 10 | 9 | 8 | 5 | 6 |

0

1

2

3

4

DFS-дерево

0

4

3

2

1

**Алгоритм топологической сортировки**

0

1

2

3

4

1G 10B

2 G 9 B

3 G 8 B

4 G 7 B

5 G 6 B

0

1

2

3

4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 3 | 2 | 1 | 0 |

**Задание 2.** Осуществить программную реализацию алгоритмов на C++. Разработать структуры **AMatrix** и **АList**  для представления ориентированного графа матричным и списковым способом. Разработать функции преобразования из одного способа представления в другой. Разработать функцию **BFS** обхода вершин графа, используя метод поиска в ширину. Продемонстрировать работу функции. Копии экрана вставить в отчет.

// --- Graph.h

//

#pragma once

#include <list>

#include "Graph.h"

namespace graph

{

struct AList;

struct AMatrix // матрица смежности

{

int n\_vertex; // количество вершин

int \*mr; // матрица

AMatrix(int n); // создать нулевую матрицу n\*n

AMatrix(int n, int mr[]); // создать матрицу n\*n и

AMatrix(const AMatrix& am); // создать подобную матрицу

AMatrix(const AList& al); // создать матрицу из спискового

void set(int i, int j, int r); // записать mr[i,j] = r

int get(int i, int j)const; // элемент mr[i,j]

};

struct AList // списки смежности

{

int n\_vertex; // количество вершин

std::list<int> \*mr; // массив списков

void create(int n); // создать массив пустых списков

AList(int n); // создать массив пустых списков

AList(int n, int mr[]); // создать списковое представление

AList(const AMatrix& am); // создать списковое представление

AList(const AList& al); // создать подобную структуру

void add(int i, int j); // добавить в i-ый список

int size(int i) const; // размер i-го списка

int get(int i, int j)const; // j-ый элемент i-го списка

};

};

// --- Graph.cpp

//

#include "stdafx.h"

#include "Graph.h"

namespace graph

{

AMatrix::AMatrix(int n)

{

this->n\_vertex = n;

this->mr = new int[this->n\_vertex\*this->n\_vertex];

for (int i = 0; i < n\*n; i++)mr[i] = 0;

};

AMatrix::AMatrix(int n, int mr[])

{

this->n\_vertex = n;

this->mr = mr;

};

AMatrix::AMatrix(const AMatrix& am)

{

this->n\_vertex = am.n\_vertex;

this->mr = new int[this->n\_vertex\*this->n\_vertex];

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < this->n\_vertex; j++)

this->set(i, j, am.get(i, j));

};

AMatrix::AMatrix(const AList& al)

{

this->n\_vertex = al.n\_vertex;

this->mr = new int[this->n\_vertex \* this->n\_vertex];

for (int k = 0; k < this->n\_vertex\*this->n\_vertex; k++)mr[k] = 0;

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < al.size(i); j++) this->set(i, al.get(i, j), 1);

};

void AMatrix::set(int i, int j, int r) { this->mr[i\*this->n\_vertex + j] = r; };

int AMatrix::get(int i, int j)const

{

return this->mr[i\*this->n\_vertex + j];

};

void AList::create(int n)

{

this->mr = new std::list<int>[this->n\_vertex = n];

};

AList::AList(int n) { create(n); }

AList::AList(const AMatrix& am)

{

this->create(am.n\_vertex);

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < this->n\_vertex; j++)

if (am.get(i, j) != 0) this->add(i, j);

};

AList::AList(const AList& al)

{

this->create(al.n\_vertex);

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < al.size(i); j++) this->add(i, al.get(i, j));

};

AList::AList(int n, int mr[])

{

this->create(n);

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < this->n\_vertex; j++)

if (mr[i\*this->n\_vertex + j] != 0) this->add(i, j);

};

void AList::add(int i, int j) { this->mr[i].push\_back(j); };

int AList::size(int i) const { return (int)this->mr[i].size(); };

int AList::get(int i, int j)const

{

std::list<int>::iterator rc = this->mr[i].begin();

for (int k = 0; k < j; k++) rc++;

return (int)\*rc;

};

};

// ---BFS.h

//

#pragma once

#include "Graph.h"

#include <queue>

struct BFS // breadth-first search поиск в ширину (связный граф)

{

const static int INF = 0x7fffffff;

const static int NIL = -1;

enum Color { WHITE, GRAY, BLACK }; //

const graph::AList \*al; // исходный граф

Color \*c; // цвет вершины

int \*d; // расстояние до вершины

int \*p; // предшествующая вершина

std::queue<int> q; // очередь

BFS(const graph::AList& al, int s);

BFS(const graph::AMatrix& am, int s);

void init(const graph::AList& al, int s);

int get(); // получить следующую вершину

};

// ---BFS.cpp

//

#include "stdafx.h"

#include "BFS.h"

void BFS::init(const graph::AList& al, int s)

{

this->al = &al;

this->c = new Color[this->al->n\_vertex];

this->d = new int[this->al->n\_vertex];

this->p = new int[this->al->n\_vertex];

for (int i = 0; i < this->al->n\_vertex; i++)

{

this->c[i] = WHITE;

this->d[i] = INF;

this->p[i] = NIL;

};

this->c[s] = GRAY;

this->q.push(s);

};

BFS::BFS(const graph::AList& al, int s) { this->init(al, s); };

BFS::BFS(const graph::AMatrix& am, int s)

{

this->init(\*(new graph::AList(am)), s);

};

int BFS::get()

{

int rc = NIL, v = NIL;

if (!this->q.empty())

{

rc = this->q.front();

for (int j = 0; j < this->al->size(rc); j++)

if (this->c[v = this->al->get(rc, j)] == WHITE)

{

this->c[v] = GRAY;

this->d[v] = this->d[rc] + 1;

this->p[v] = rc;

this->q.push(v);

};

this->q.pop();

this->c[rc] = BLACK;

};

return rc;

}

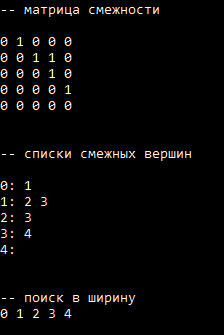


Рис. 1 – результат выполнения программы

**Задание 3.**  Разработать функцию **DFS**  обхода вершин графа, используя метод поиска глубину. Продемонстрировать работу функции. Копии экрана вставить в отчет.

// ---DFS.h

//

#pragma once

#include "Graph.h"

#include <vector>

struct DFS // depth-first search поиск в глубину

{

const static int NIL = -1;

enum Color { WHITE, GRAY, BLACK }; //

const graph::AList \*al; // исходный граф

Color \*c; // цвет вершины

int \*d; // время обнаружения

int \*f; // время завершения обработки

int \*p; // предшествующая вершина

int t; // текущее время

DFS(const graph::AList& al);

DFS(const graph::AMatrix& am);

std::vector <int> topological\_sort; //результат топологической сортировки

void visit(int v);

void init(const graph::AList& al);

int get(int i); // получить вершину

};

// ---DFS.cpp

//

#include "stdafx.h"

#include "DFS.h"

#define NINF 0x80000000

#define INF 0x7fffffff

void DFS::init(const graph::AList& al)

{

this->al = &al;

this->c = new Color[this->al->n\_vertex];

this->d = new int[this->al->n\_vertex];

this->f = new int[this->al->n\_vertex];

this->p = new int[this->al->n\_vertex];

this->t = 0;

for (int i = 0; i < this->al->n\_vertex; i++)

{

this->c[i] = WHITE;

this->d[i] = this->f[i] = 0;

this->p[i] = NIL;

};

for (int i = 0; i < this->al->n\_vertex; i++)

if (this->c[i] == WHITE)

{

this->visit(i);

this->topological\_sort.push\_back(i);

}

};

DFS::DFS(const graph::AList& al) { this->init(al); };

DFS::DFS(const graph::AMatrix& am)

{

this->init(\*(new graph::AList(am)));

};

void DFS::visit(int u)

{

int v = NIL;

this->c[u] = GRAY;

this->d[u] = ++(this->t);

for (int j = 0; j < this->al->size(u); j++)

if (this->c[v = this->al->get(u, j)] == WHITE)

{

this->p[v] = u;

this->visit(v);

this->topological\_sort.push\_back(v);

}

this->c[u] = BLACK;

this->f[u] = ++(this->t);

};

int DFS::get(int i)

{

int j = 0, min1 = INF, min2 = NINF, ntx = NIL;

for (int j = 0; j <= i; j++) // iая статистика

{

for (int k = 0; k < this->al->n\_vertex; k++)

if (this->f[k] < min1 && this->f[k] > min2)

{

min1 = this->f[k]; ntx = k;

};

min2 = min1; min1 = INF;

};

return ntx;

};



Рис. 2 – результат выполнения программы

**Задание 4.** Доработайте функцию **DFS**,для выполнения топологической сортировки графа. Продемонстрировать работу функции. Копии экрана вставить в отчет.

// ---DFS.h

//

#pragma once

#include "Graph.h"

#include <vector>

struct DFS // depth-first search поиск в глубину

{

const static int NIL = -1;

enum Color { WHITE, GRAY, BLACK }; //

const graph::AList \*al; // исходный граф

Color \*c; // цвет вершины

int \*d; // время обнаружения

int \*f; // время завершения обработки

int \*p; // предшествующая вершина

int t; // текущее время

DFS(const graph::AList& al);

DFS(const graph::AMatrix& am);

std::vector <int> topological\_sort; //результат топологической сортировки

void visit(int v);

void init(const graph::AList& al);

int get(int i); // получить вершину

};

// ---DFS.cpp

//

#include "stdafx.h"

#include "DFS.h"

#define NINF 0x80000000

#define INF 0x7fffffff

void DFS::init(const graph::AList& al)

{

this->al = &al;

this->c = new Color[this->al->n\_vertex];

this->d = new int[this->al->n\_vertex];

this->f = new int[this->al->n\_vertex];

this->p = new int[this->al->n\_vertex];

this->t = 0;

for (int i = 0; i < this->al->n\_vertex; i++)

{

this->c[i] = WHITE;

this->d[i] = this->f[i] = 0;

this->p[i] = NIL;

};

for (int i = 0; i < this->al->n\_vertex; i++)

if (this->c[i] == WHITE)

{

this->visit(i);

this->topological\_sort.push\_back(i);

}

};

DFS::DFS(const graph::AList& al) { this->init(al); };

DFS::DFS(const graph::AMatrix& am)

{

this->init(\*(new graph::AList(am)));

};

void DFS::visit(int u)

{

int v = NIL;

this->c[u] = GRAY;

this->d[u] = ++(this->t);

for (int j = 0; j < this->al->size(u); j++)

if (this->c[v = this->al->get(u, j)] == WHITE)

{

this->p[v] = u;

this->visit(v);

this->topological\_sort.push\_back(v);

}

this->c[u] = BLACK;

this->f[u] = ++(this->t);

};

int DFS::get(int i)

{

int j = 0, min1 = INF, min2 = NINF, ntx = NIL;

for (int j = 0; j <= i; j++) // iая статистика

{

for (int k = 0; k < this->al->n\_vertex; k++)

if (this->f[k] < min1 && this->f[k] > min2)

{

min1 = this->f[k]; ntx = k;

};

min2 = min1; min1 = INF;

};

return ntx;

};



Рис. 3 – результат работы программы

**Вывод:** в данной работе были освоены общие принципы представления графов, а именно с помощью списков и матриц. Были освоены сущности программной реализации для представления графов. Также были изучены алгоритмы обходов графов, таки как поиск в ширину, глубину и глубину.