Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ

УНИВЕРСИТЕТ»

Отчёт по лабораторной работе №5

«**Алгоритмы на графах**»

Студентка: Змитревич Д.А.

ФИТ 2 курс 9 группа

Преподаватель: Барковский Е.В.

Минск 2022

**ЦЕЛЬ РАБОТЫ:** 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. Оформить отчет, включив в него **каждый** шаг выполнения алгоритмов.

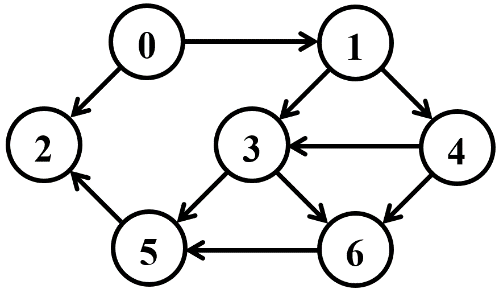


Рисунок 5.1 – Ориентированный граф G

Массивы, *поиск в ширину*:

Q – для промежуточного хранения вершин (очередь)

С – массив окраски вершин

D – массив расстояний

Р – массив предшествующих вершин

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Шаг 1.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | Q | 0 |  |  |  |  |  |  | | C | G | W | W | W | W | W | W | | D | 0 |  |  |  |  |  |  | | P | N | N | N | N | N | N | N | |
|  | Шаг 2.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | Q | 1 | 2 |  |  |  |  |  | | C | B | G | G | W | W | W | W | | D | 0 | 1 | 1 |  |  |  |  | | P | N | 0 | 0 | N | N | N | N | |
|  | Шаг 3.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | Q | 3 | 4 |  |  |  |  |  | | C | B | B | G | G | G | W | W | | D | 0 | 1 | 1 | 2 | 2 |  |  | | P | N | 0 | 0 | 1 | 1 | N | N | |
|  | Шаг 4.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | Q | 5 | 6 |  |  |  |  |  | | C | B | B | B | B | G | G | G | | D | 0 | 1 | 1 | 2 | 2 | 3 | 3 | | P | N | 0 | 0 | 1 | 1 | 3 | 3 | |
|  | Шаг 5.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | Q |  |  |  |  |  |  |  | | C | B | B | B | B | B | B | B | | D | 0 | 1 | 1 | 2 | 2 | 3 | 3 | | P | N | 0 | 0 | 1 | 1 | 3 | 3 | |
|  | Готовое BFS-дерево. |

Массивы, *поиск в глубину*:

С – массив окраски вершин

D – массив расстояний

Р – массив предшествующих вершин

F – время окраски в черный цвет

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изображение выглядит как текст, часы  Автоматически созданное описание | Шаг 1.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | С | G | W | W | W | W | W | W | | D | 1 |  |  |  |  |  |  | | P | N | N | N | N | N | N | N | | F | N | N | N | N | N | N | N | |
|  | Шаг 2.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | С | G | G | W | W | W | W | W | | D | 1 | 2 |  |  |  |  |  | | P | N | 0 | N | N | N | N | N | | F | N | N | N | N | N | N | N | |
|  | Шаг 3.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | С | G | G | G | W | W | W | W | | D | 1 | 2 | 3 |  |  |  |  | | P | N | 0 | 1 | N | N | N | N | | F | N | N | N | N | N | N | N | |
|  | Шаг 4.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | С | G | G | G | G | W | W | W | | D | 1 | 2 | 3 | 4 |  |  |  | | P | N | 0 | 1 | 3 | N | N | N | | F | N | N | N | N | N | N | N | |
|  | Шаг 5.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | С | G | G | G | G | G | W | W | | D | 1 | 2 | 3 | 4 | 5 |  |  | | P | N | 0 | 1 | 3 | 5 | N | N | | F | N | N | N | N | N | N | N |   Шаг 6.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | С | G | G | G | G | B | W | W | | D | 1 | 2 | 3 | 4 | 5 | 6 |  | | P | N | 0 | 1 | 3 | 5 | 5 | N | | F | N | N | N | N | N | N | 6 |   Шаг 7.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | С | G | G | G | B | B | G | W | | D | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | P | N | 0 | 1 | 3 | 5 | 5 | 3 | | F | N | N | N | N | N | 7 | 6 |   Шаг 8.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | С | G | G | G | B | B | B | W | | D | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | P | N | 0 | 1 | 3 | 5 | 5 | 3 | | F | N | N | N | N | 8 | 7 | 6 |   Шаг 9.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | С | G | G | B | B | B | B | G | | D | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | P | N | 0 | 1 | 3 | 5 | 5 | 3 | | F | N | N | N | 9 | 8 | 7 | 6 |   Шаг 10.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | С | G | G | B | B | B | B | B | | D | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | P | N | 0 | 1 | 3 | 5 | 5 | 3 | | F | N | N | 10 | 9 | 8 | 7 | 6 |   Шаг 11.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | С | G | B | B | B | B | B | B | | D | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | P | N | 0 | 1 | 3 | 5 | 5 | 3 | | F | N | 11 | 10 | 9 | 8 | 7 | 6 |   Шаг 12.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | С | B | B | B | B | B | B | B | | D | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | P | N | 0 | 1 | 3 | 5 | 5 | 3 | | F | 12 | 11 | 10 | 9 | 8 | 7 | 6 | |
|  | Готовое DFS-дерево. |

*Топологическая сортировка* – это процедура упорядочивания вершин ориентированного графа, не имеющего циклов.

В результате топологической сортировки для вершин графа определяется такой порядок, что если их расположить на рисунке в соответствии с этим порядком сверху вниз, то дуги будут направлены только от верхних вершин к нижним.

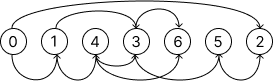


Рисунок 5.2 – Топологическая сортировка

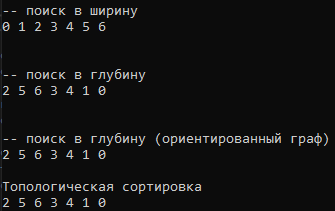


Рисунок 5.2 – Результат проверки сортировок

**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.cpp

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.cpp

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;

}

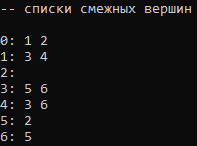
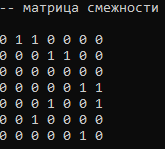
 

Рисунок 5.4 и 5.5 – Результат проверки задания

**Задание 3-4**. Разработать функцию **DFS** обхода вершин графа, используя метод поиска глубину, и для выполнения топологической сортировки графа. Продемонстрировать работу функции. Копии экрана вставить в отчет.

// DFS.cpp

#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++)

{

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;

};

**Вывод**: в ходе данной работы были освоена сущность способов представления графов, реализация алгоритмов поиска в ширину и глубину и топологической сортировки.