Белорусский государственный технологический университет

Кафедра Программной инженерии

**Отчет по лабораторной работе №5**

**АЛГОРИТМЫ НА ГРАФАХ**

**Вариант 10**

Выполнила: Пунько Алина

ФИТ 2 курс, 6 группа

Проверил: Буснюк Н. Н.

Минск 2019

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

|  |  |
| --- | --- |
| 10 |  |

**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 алгоритмах.

**Ниже приведена демонстрация работы поиска в глубину.**

Текущее состояние алгоритма хранится в следующих структурах памяти:

**C** – массив окраски вершин,

**D** – время окраски в серый цвет,

**P** – массив предшествующих вершин,

**F** – время окраски в чёрный цвет.

Кроме того, используется переменная t, текущее значение которой – номер шага алгоритма.

По условию, граф имеет 5 вершин, пронумерованных начиная с нуля. В качестве стартовой вершины выбрана вершина с номером 0.

Шаг 1. Начинаем с нулевой вершины и красим её. Находим смежные вершины вершине 0. Это вершина 1.

1

0

2

3

4

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | G | W | W | W | W |
| D | 1 | I | I | I | I |
| P | N | N | N | N | N |
| F | 0 | 0 | 0 | 0 | 0 |

Шаг 2: затем переходим в 1-ю из 0-й и красим её. Находим смежные вершины вершине 1. Это вершина 2.

1

0

2

3

4

|  |  |
| --- | --- |
| 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 |

Шаг 3: потом переходим во 2-ю из 1-й и красим её. Находим смежные вершины вершине 2. Это вершина 3.

1

0

2

3

4

|  |  |
| --- | --- |
| 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 |

Шаг 4: потом переходим во 3-ю из 2-й и красим её. Ищем смежные вершины вершине 3. Это вершины 0 и 4.

0

1

2

4

3

|  |  |
| --- | --- |
| 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 |

Шаг 5: потом переходим во 4-ю из 3-й и красим её. Находим смежные вершины вершине 4. Это вершины 0 и 4.

1

0

2

3

4

|  |  |
| --- | --- |
| 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 |

Шаг 6. Белых вершин, смежный с 4 вершиной, нет, следовательно, мы погрузились в глубину. Закрашиваем 4 вершину в чёрный цвет.

1

0

2

3

4

|  |  |
| --- | --- |
| 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 |

Шаг 7. На предыдущем и последующих шагах массивы D и P – не изменяются; заполняется массив F и изменяется массив C.

1

0

2

3

4

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | G | G | G | B | B |
| D | 1 | 2 | 3 | 4 | 5 |
| P | N | 0 | 1 | 2 | 3 |
| F | 0 | 0 | 0 | 7 | 6 |

Шаг 8. Окрашиваем вершину 2 в чёрный цвет.

1

0

2

3

4

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | G | G | B | B | B |
| D | 1 | 2 | 3 | 4 | 5 |
| P | N | 0 | 1 | 2 | 3 |
| F | 0 | 0 | 8 | 7 | 6 |

Шаг 9. Окрашиваем вершину 1 в чёрный цвет.

1

0

2

3

4

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | G | B | B | B | B |
| D | 1 | 2 | 3 | 4 | 5 |
| P | N | 0 | 1 | 2 | 3 |
| F | 0 | 9 | 8 | 7 | 6 |

Шаг 10. Последний шаг: окрашиваем 1 вершину в чёрный цвет и проверяем, что смежный ей вершин серого цвета нет. Значит алгоритм закончил свою работу.

1

0

2

3

4

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | B | B | B | B | B |
| D | 1 | 2 | 3 | 4 | 5 |
| P | N | 0 | 1 | 2 | 3 |
| F | 10 | 9 | 8 | 7 | 6 |

**Ответ:** DFS-дерево имеет вид:

1

0

2

3

4

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

**Ниже представлен результат работы поиска в ширину.**

Текущее состояние алгоритма хранится в следующих структурах памяти:

**Q** – очередь вершин,

**C** – массив окраски вершин,

**D** – массив расстояний,

**P** – массив предшествующих вершин.

По условию, граф имеет 5 вершин, пронумерованных начиная с нуля. В качестве стартовой вершины выбрана вершина с номером 0.

Шаг 1. Начинаем с нулевой вершины, помещаем ее в очередь и красим её.

1

0

2

3

4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Q | 0 |  |  |  |  |
| C | G | W | W | W | W |
| D | 0 | I | I | I | I |
| P | N | N | N | N | N |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Q | 1 |  |  |  |  |
| C | B | G | W | W | W |
| D | 0 | 1 | I | I | I |
| P | N | 0 | N | N | N |

Шаг 2. Извлекаем из очереди 0-ую вершину и закрашиваем в black (B) цвет; помещаем 1-ую вершину в очередь.

1

0

2

3

4

Шаг 3. Извлекаем из очереди 1-ую вершину и закрашиваем в black (B) цвет; помещаем 2-ую вершину в очередь.

1

0

2

3

4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Q | 2 |  |  |  |  |
| C | B | B | G | W | W |
| D | 0 | 1 | 2 | I | I |
| P | N | 0 | 1 | N | N |

Шаг 4. Извлекаем из очереди 2-ую вершину и закрашиваем в black (B) цвет; помещаем 3-ую вершину в очередь.

1

0

2

3

4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Q | 3 |  |  |  |  |
| C | B | B | B | G | W |
| D | 0 | 1 | I | 3 | I |
| P | N | 0 | 1 | 2 | N |

Шаг 5. Извлекаем из очереди 3-юю вершину и закрашиваем в black (B) цвет; помещаем 4-ую вершину в очередь.

1

0

2

3

4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Q | 4 |  |  |  |  |
| C | B | B | B | B | G |
| D | 0 | 1 | 2 | 3 | 4 |
| P | N | 0 | 1 | 2 | 3 |

Шаг 6: последний шаг алгоритма. Извлекаем последний элемент из очереди, очередь пуста, все вершины окрашиваются в В-цвет. Извлекаем последнюю 4-ую вершину, проверяем на смежность с другими вершинами.

1

0

2

3

4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Q | 5 |  |  |  |  |
| C | B | B | B | B | B |
| D | 0 | 1 | 2 | 3 | 4 |
| P | N | 0 | 1 | 2 | 3 |

Далее если мы попытаемся зайти в какую-либо вершину, то она уже будет закрашена и нас выкинет.

**Ответ:** Все вершины чёрного цвета, соответственно, алгоритм закончил свою работу. В результате получили вот такое BFS-дерево:

1

0

2

3

4

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

**Алгоритм топологической сортировки**

Исходный граф:

1

0

2

3

4

По условию, граф имеет 5 вершин, пронумерованных начиная с нуля. В качестве стартовой вершины выбрана вершина с номером 0.

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

При реализации топологической сортировки с помощью алгоритма поиска в глубину используется массив меток вершин, с помощью которого моделируется удаление вершин из графа и сохраняются новые номера вершин.

Шаг 1. На первом шаге окрашиваем 0-ую вершину в серый цвет.

1/10

1/10

2/9

1

0

2

3

3/8

4

4/7

5/6

Формируем очередь по мере окрашивания вершин в чёрный цвет: 4 3 2 1 0

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 | | Graph.cpp |
| 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  }; | | 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  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  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;  } | |

Ниже на рисунке 1 демонстрируется результат работы поиска в ширину.

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

|  |  |
| --- | --- |
| DFS.h | DFS.cpp |
| 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  }; | 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;  }; |

На рисунке 1 продемонстрирован результат работы поиска в глубину.

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

Топологические сортировки для матрицы и списка смежностей реализованы как методы классов и их код был представлен выше. Ниже на рисунке 1 представлен результат работы:

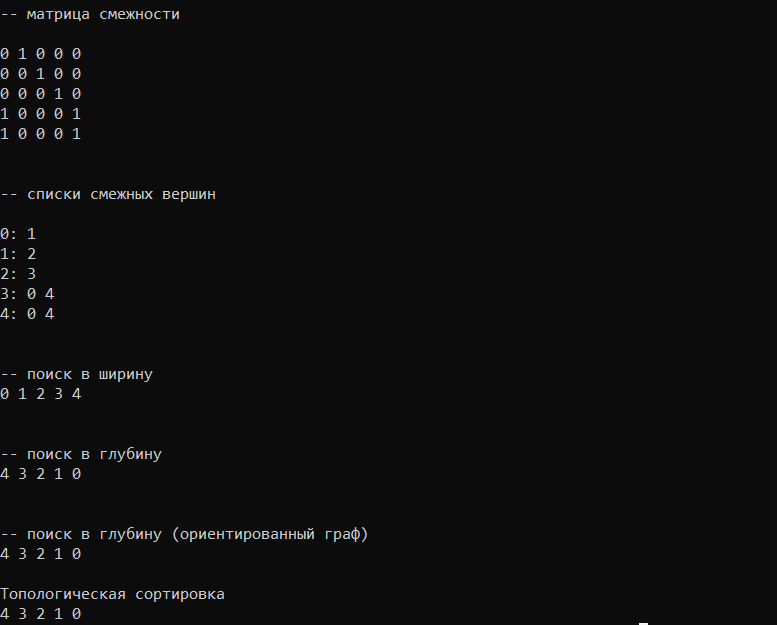


Рисунок 1. Результат работы