**Лабораторная работа 6. АЛГОРИТМЫ НА ГРАФАХ**

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

**ЗАДАНИЕ ДЛЯ ВЫПОЛНЕНИЯ:**

***Задание 1.*** Ориентированный граф **G** взять в соответствии с вариантом. Представить его в отчете в виде матрицы смежности, матрицы инцидентности, списка смежных вершин.

|  |  |
| --- | --- |
| 2 | C:\Users\lenovo\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\изображение_2024-04-28_113241537.jpeg |

Матрица смежности:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **0** | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| **3** | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| **4** | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| **5** | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Список смежностей:

0: {2}

1: ∅

2: {3,5}

3: {1,4,5,6}

4: {1,6}

5: {6}

6: ∅

Матрица инцидентности:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **0** | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| **2** | 1 | -1 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 1 | 0 | 0 | -1 | -1 | -1 | -1 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | -1 | -1 |
| **5** | 0 | 0 | 1 | -1 | 1 | 0 | 0 | 0 | 0 | 0 |
| **6** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |

***Задание 2.*** 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.

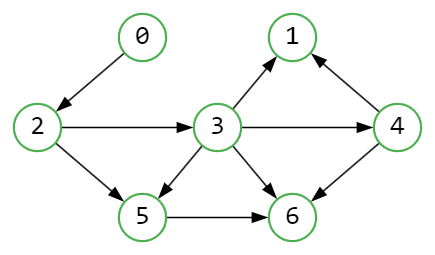
Q – очередь вершин(FIFO), промежуточное хранение. Номера помещаются в порядке обнаружения вершин. Первый шаг – стартовая вершина. Последний – пусто.

C – массив окраски вершин(W – не посещалась, G – посещалась, B – факт посещения).

D – расстояний. Первый шаг – ноль для стартовой вершины и бесконечности для остальных.

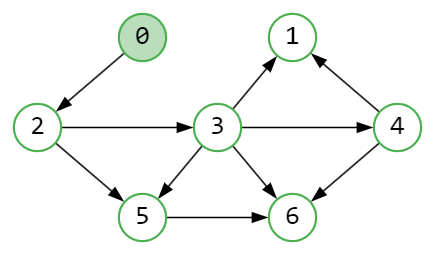
P – предшествующие.

**Поиск в ширину**



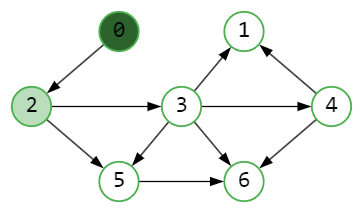
Шаг 1:

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



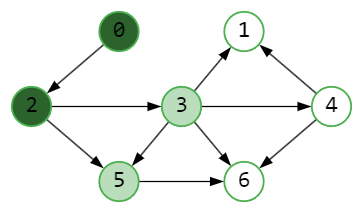
Шаг 2:

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



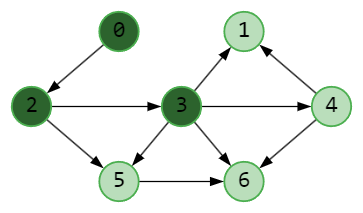
Шаг 3:

|  |  |  |
| --- | --- | --- |
| **Q** | 3 | 5 |
| **C** | B | W | B | G | W | G | W |
| **D** | 0 | | | 1 | 2 | | | 2 | | |
| **P** | N | N | 0 | 2 | N | 2 | N |



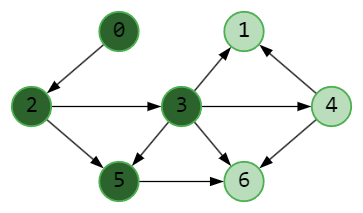
Шаг 4:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Q** | 5 | 1 | 4 | 6 |
| **C** | B | G | B | B | G | G | G |
| **D** | 0 | 3 | 1 | 2 | 3 | 2 | 3 |
| **P** | N | 3 | 0 | 2 | 3 | 2 | 3 |



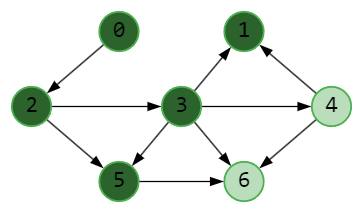
Шаг 5:

|  |  |  |  |
| --- | --- | --- | --- |
| **Q** | 1 | 4 | 6 |
| **C** | B | G | B | B | G | B | G |
| **D** | 0 | 3 | 1 | 2 | 3 | 2 | 3 |
| **P** | N | 3 | 0 | 2 | 3 | 2 | 3 |



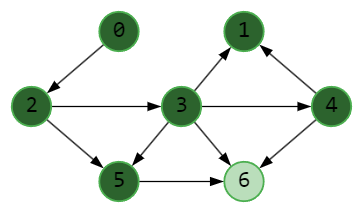
Шаг 6:

|  |  |  |
| --- | --- | --- |
| **Q** | 4 | 6 |
| **C** | B | B | B | B | G | B | G |
| **D** | 0 | 3 | 1 | 2 | 3 | 2 | 3 |
| **P** | N | 3 | 0 | 2 | 3 | 2 | 3 |



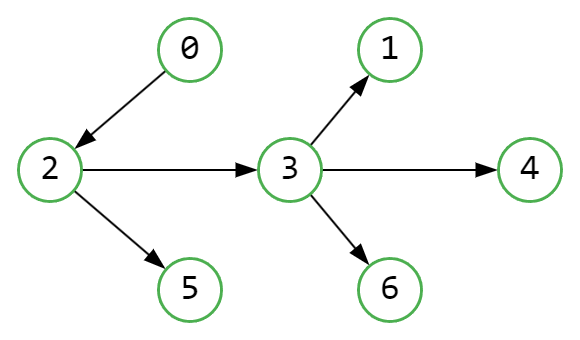
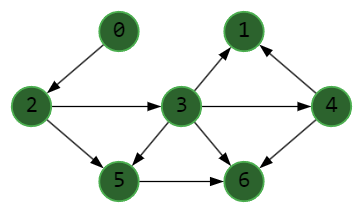
Шаг 7:

|  |  |
| --- | --- |
| **Q** | 6 |
| **C** | B | B | B | B | B | B | G |
| **D** | 0 | 3 | 1 | 2 | 3 | 2 | 3 |
| **P** | N | 3 | 0 | 2 | 3 | 2 | 3 |



Шаг 8:

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



В итоге получили обход в ширину по следующим вершинам: 0,2,3,5,1,4,6

**Поиск в глубину**

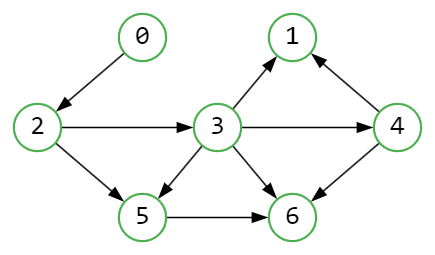
С – массив окраски вершин.

P – массив предшествующих.

D – шаг окраски в серый.

F – фиксация окраски в черный.

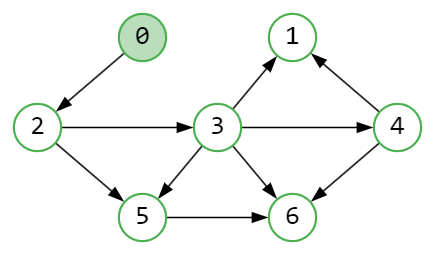
t = 0



Шаг 1:

t = 1

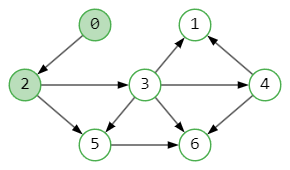
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | W | W | W | W | W | W |
| D | 1 | | | | | | | | | | | | |
| P | N | N | N | N | N | N | N |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



Шаг 2:

t = 2

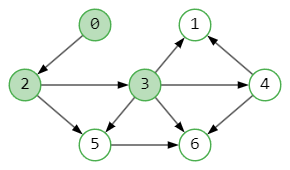
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | W | G | W | W | W | W |
| D | 1 | | | 2 | | | | | | | | |
| P | N | N | 0 | N | N | N | N |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



Шаг 3:

t = 3

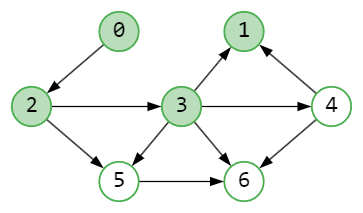
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | W | G | G | W | W | W |
| D | 1 | | | 2 | 3 | | | | | | |
| P | N | N | 0 | 2 | N | N | N |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



Шаг 4:

t = 4

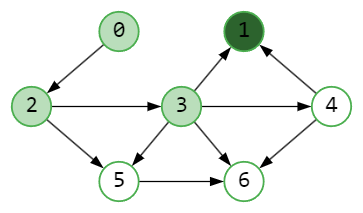
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | G | G | W | W | W |
| D | 1 | 4 | 2 | 3 | | | | | | |
| P | N | 3 | 0 | 2 | N | N | N |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



Шаг 5:

t = 5

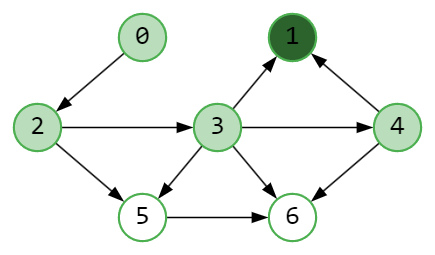
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | G | G | W | W | W |
| D | 1 | 4 | 2 | 3 | | | | | | |
| P | N | 3 | 0 | 2 | N | N | N |
| F | 0 | 5 | 0 | 0 | 0 | 0 | 0 |



Шаг 6:

t = 6

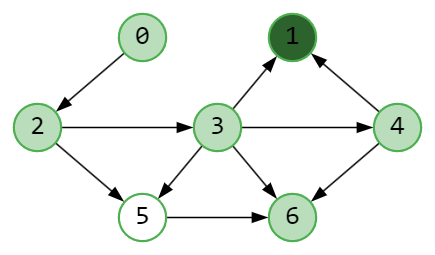
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | G | G | G | W | W |
| D | 1 | 4 | 2 | 3 | 6 | | | | |
| P | N | 3 | 0 | 2 | 3 | N | N |
| F | 0 | 5 | 0 | 0 | 0 | 0 | 0 |



Шаг 7:

t = 7

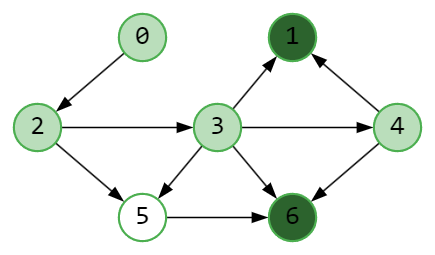
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | G | G | G | W | G |
| D | 1 | 4 | 2 | 3 | 5 | | | 7 |
| P | N | 3 | 0 | 2 | 3 | N | 4 |
| F | 0 | 6 | 0 | 0 | 0 | 0 | 0 |



Шаг 8:

t = 8

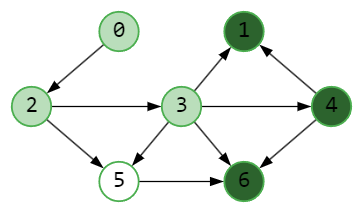
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | G | G | G | W | B |
| D | 1 | 4 | 2 | 3 | 5 | | | 7 |
| P | N | 3 | 0 | 2 | 3 | N | 4 |
| F | 0 | 6 | 0 | 0 | 0 | 0 | 8 |



Шаг 9:

t = 9

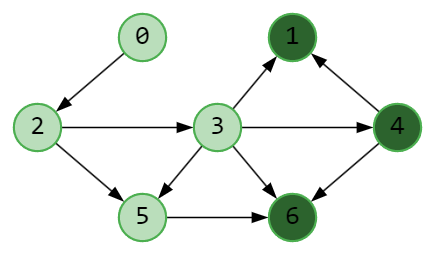
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | G | G | B | W | B |
| D | 1 | 4 | 2 | 3 | 5 | | | 7 |
| P | N | 3 | 0 | 2 | 3 | N | 4 |
| F | 0 | 6 | 0 | 0 | 9 | 0 | 8 |



Шаг 10:

t = 10

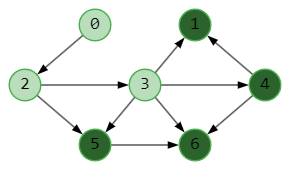
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | G | G | B | G | B |
| D | 1 | 4 | 2 | 3 | 5 | 10 | 7 |
| P | N | 3 | 0 | 2 | 3 | 3 | 4 |
| F | 0 | 6 | 0 | 0 | 9 | 0 | 8 |



Шаг 11:

t = 11

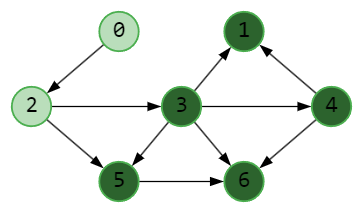
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | G | G | B | B | B |
| D | 1 | 4 | 2 | 3 | 5 | 10 | 7 |
| P | N | 3 | 0 | 2 | 3 | 3 | 4 |
| F | 0 | 6 | 0 | 0 | 9 | 11 | 8 |



Шаг 12:

t = 12

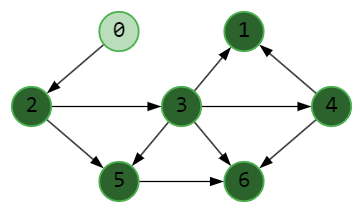
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | G | B | B | B | B |
| D | 1 | 4 | 2 | 3 | 5 | 10 | 7 |
| P | N | 3 | 0 | 2 | 3 | 3 | 4 |
| F | 0 | 6 | 0 | 12 | 9 | 11 | 8 |



Шаг 13:

t = 13

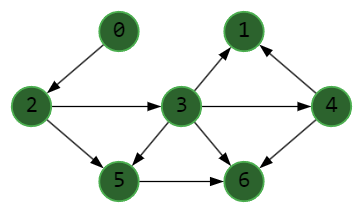
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | B | B | B | B | B |
| D | 1 | 4 | 2 | 3 | 5 | 10 | 7 |
| P | N | 3 | 0 | 2 | 3 | 3 | 4 |
| F | 0 | 6 | 13 | 12 | 9 | 11 | 8 |

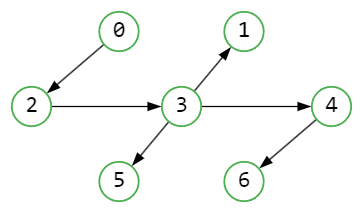


Шаг 14:

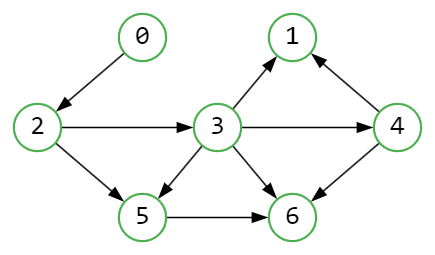
t = 14

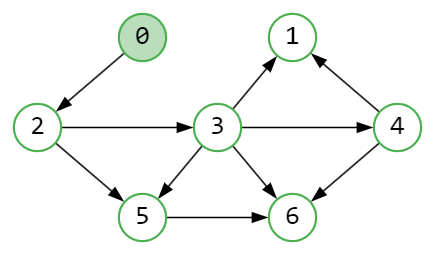
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | B | B | B | B | B | B | B |
| D | 1 | 4 | 2 | 3 | 5 | 10 | 7 |
| P | N | 3 | 0 | 2 | 3 | 3 | 4 |
| F | 14 | 6 | 13 | 12 | 9 | 11 | 8 |

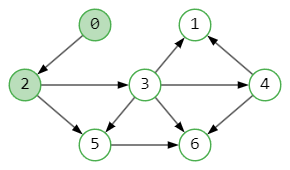


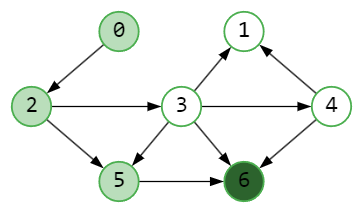
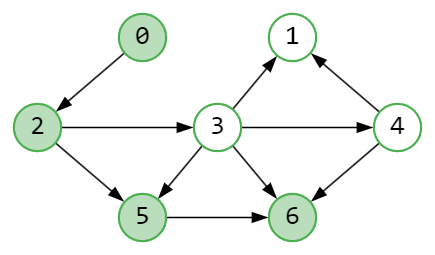
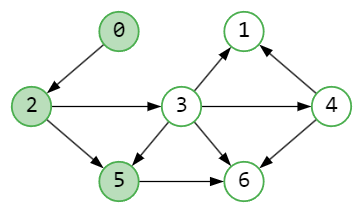


**Топологическая сортировка**

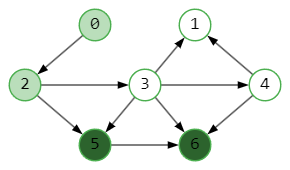
****



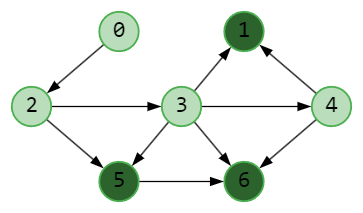
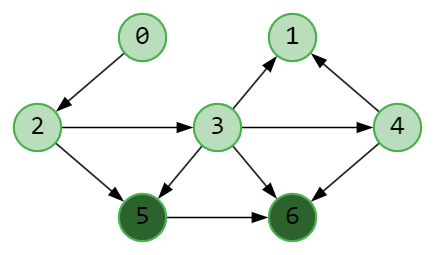
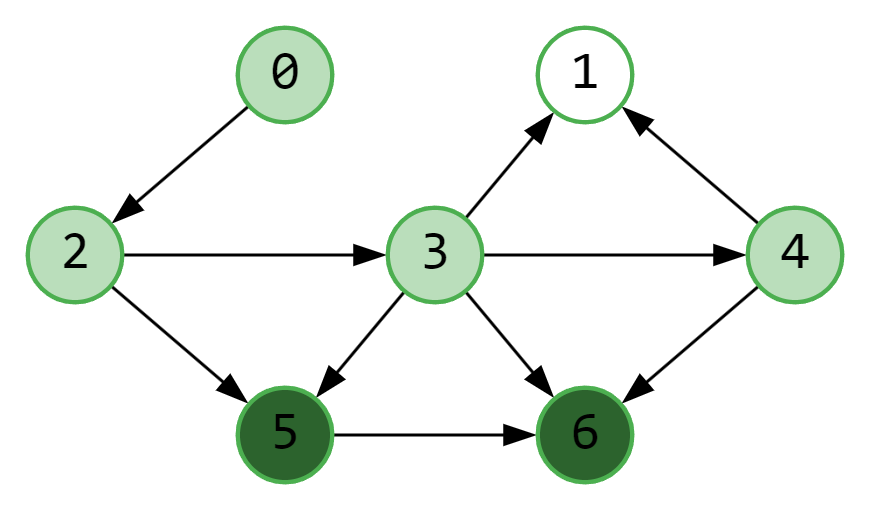
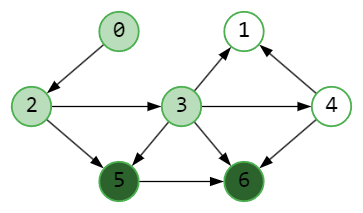
****



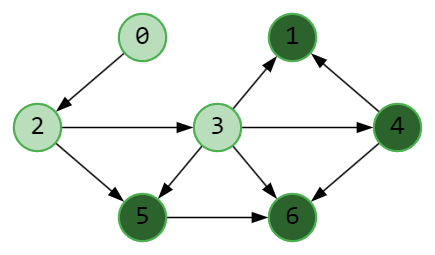
|  |
| --- |
| 6 |



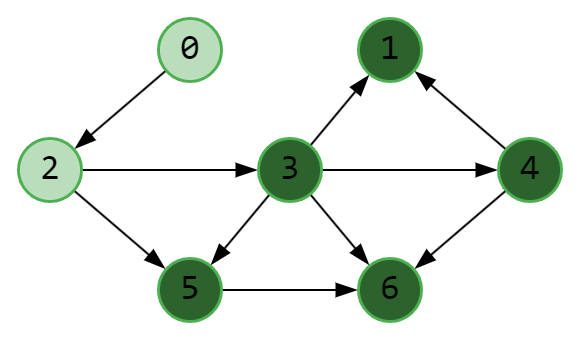
|  |  |
| --- | --- |
| 5 | 6 |



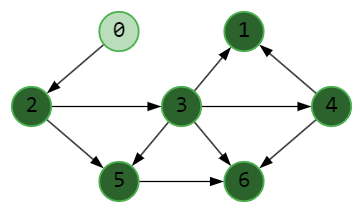
|  |  |  |
| --- | --- | --- |
| 1 | 5 | 6 |



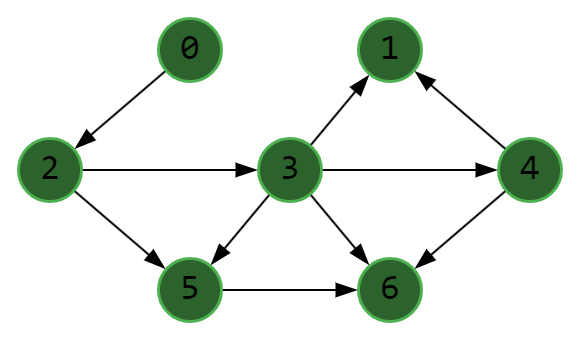
|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 1 | 5 | 6 |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 4 | 1 | 5 | 6 |

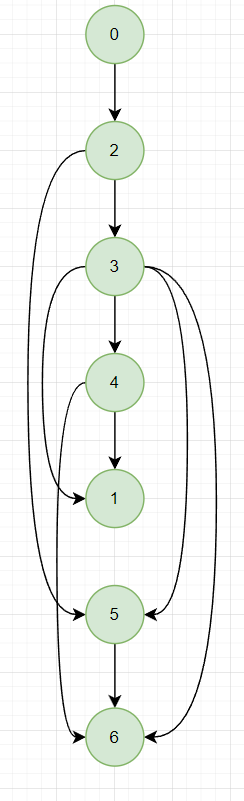


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 3 | 4 | 1 | 5 | 6 |



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 2 | 3 | 4 | 1 | 5 | 6 |

|  |
| --- |
| 0 |
| 2 |
| 3 |
| 4 |
| 1 |
| 5 |
| 6 |



***Задание 3.*** 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.

|  |
| --- |
| #pragma once  #include "Graph.h"  #include <queue>  struct BFS  {  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  }; |

Листинг 1 – BFS.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;  } |

Листинг 2 – BFS.cpp

***Задание 4.*** Разработать функцию **DFS**  обхода вершин графа, используя метод поиска глубину. Продемонстрировать работу функции. Копии экрана вставить в отчет.

|  |
| --- |
| #pragma once  #include "Graph.h"  #include <vector>  struct DFS  {  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  }; |

Листинг 3 – DFS.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++)  {  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;  }; |

Листинг 4 – DFS.cpp

***Задание 5.*** Доработайте функцию **DFS**,для выполнения топологической сортировки графа. Продемонстрировать работу функции. Копии экрана вставить в отчет.

|  |
| --- |
| 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 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;  } |

Листинг 5 – топологическая сортировка

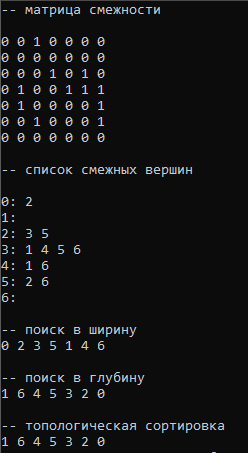
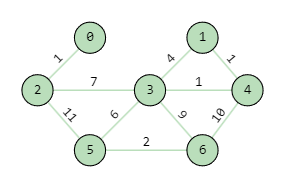


Рисунок 1 – результат работы программы

***Задание 6.*** По графу, соответствующему варианту составить минимальное остовное дерево по алгоритму Прима. Шаги построения отразить в отчете.

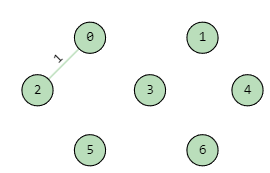


Шаг 1:

Предположим, что в множество U пока только входит вершина 0, нужно найти ребро с минимальной стоимостью, которое соединяет вершину 0 с вершиной, которая в множество U не входит.

U = {0}

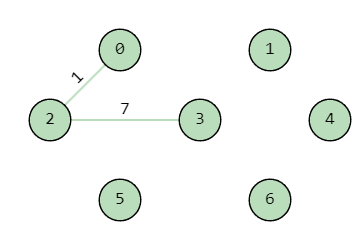
Вершина 0 соединена с вершинами 2. Вес ребра равен 1. Тогда, вершину 2 включаем в множество U, а ребро 0-2 — в множество T.



Шаг 2:

U = {0, 2}

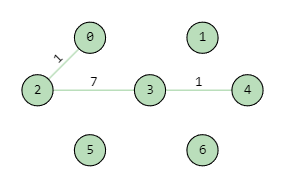
Вершины 0 и 2 соединены с вершинами 3 и 5. Веса их ребер равно 7 и 11 соответственно. Из этих ребер минимальной стоимостью обладает ребро 2-3. Тогда, вершину 3 включаем в множество U, а ребро 2-3 — в множество T.



Шаг 3:

U = {0, 2, 3}

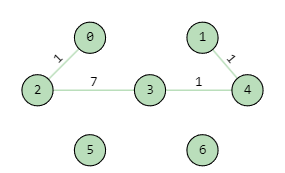
Вершины 0, 2 и 3 соединены с вершинами 1, 5(2-5), 4, 6, 5(3-5). Веса их ребер равно 4, 11, 1, 9 и 6 соответственно. Из этих ребер минимальной стоимостью обладает ребро 3-4 (вес = 1). Тогда, вершину 4 включаем в множество U, а ребро 3-4 — в множество T.



Шаг 4:

U = {0, 2, 3, 4}

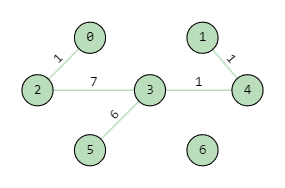
Вершины 0, 2, 3 и 4 соединены с вершинами 1(3-1),1(4-1), 5(2-5), 5(3-5), 6(3-6), 6(4-6). Веса их ребер равно 4,1, 11, 6, 9 и 10 соответственно. Из этих ребер минимальной стоимостью обладает ребро 4-1 (вес = 1). Тогда, вершину 1 включаем в множество U, а ребро 4-1 — в множество T.



Шаг 5:

U = {0, 2, 3, 4, 1}

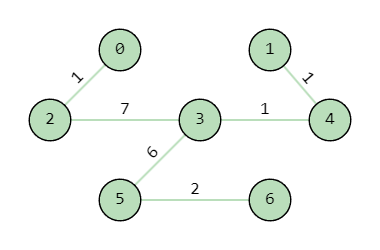
Вершины 0, 2, 3, 4 и 1 соединены с вершинами 5(2-5),5(3-5), 6(3-6), 6(4-6) .Веса их ребер равно 11, 6, 9, 10 соответственно. Из этих ребер минимальной стоимостью обладает ребро 3-5 (вес = 6). Тогда, вершину 5 включаем в множество U, а ребро 3-5 — в множество T.



Шаг 6:

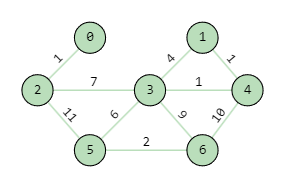
U = {0, 2, 3, 4, 1, 5}

Оставшаяся вершина 6 соединяется с вершиной 5, 3 и 4, их веса равны 2, 9 и 10 соответственно. Тогда, вершину 6 включаем в множество U, а ребро 5-6 — в множество T.



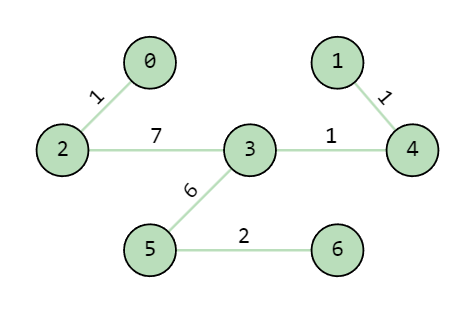
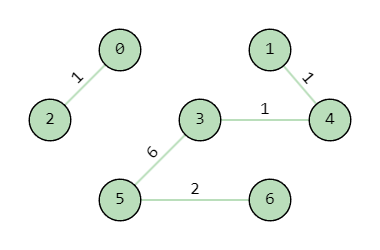
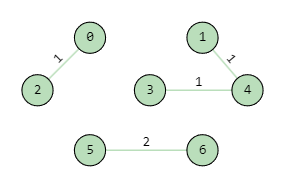
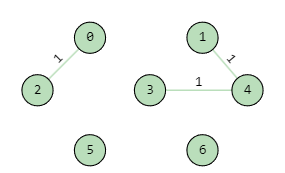
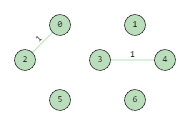
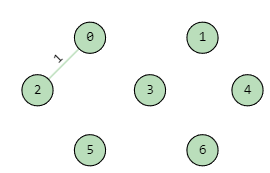
Вес минимального остовного дерева: 18

***Задание 7.*** По графу, соответствующему варианту составить минимальное остовное дерево по алгоритму Крускала. Шаги построения отразить в отчете



Шаг 1:

Из всех ребер выбираем ребро с минимальной стоимостью (весом) и включаем это ребро в остовное дерево. Повторяем так далее, однако смотрим на то, чтобы следующее ребро соединяло вершины из разных компонент.



Вес минимального остовного дерева: 18