Белорусский государственный технологический университет

факультет информационных технологий

кафедра информационных систем и технологий

**Отчёт по предмету**

**«Математическое программирование»**

**Вариант 12**

Немкович Анастасия Вадимовна

ФИТ, 2 курс, группа 1/1

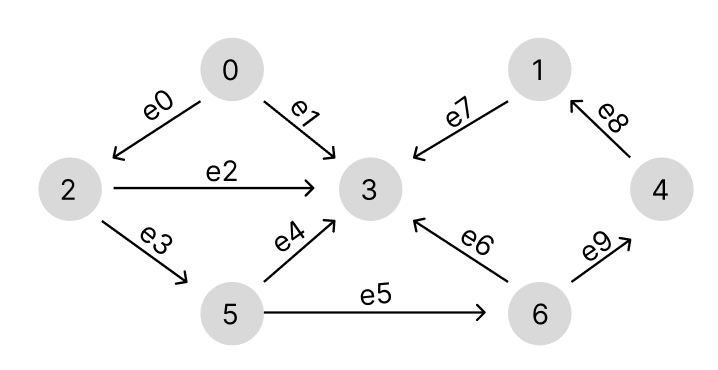
Минск 2024

**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 1.***

|  |  |
| --- | --- |
| Вариант | Граф G |
| 12 |  |



Матрица смежности:

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

Матрица инцидентности:

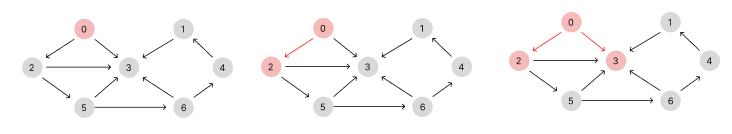
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **е0** | **е1** | **е2** | **е3** | **е4** | **е5** | **е6** | **е7** | **е8** | **е9** |
| **0** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | -1 | 0 |
| **2** | -1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | -1 | -1 | 0 | -1 | 0 | -1 | -1 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | -1 |
| **5** | 0 | 0 | 0 | -1 | 1 | 1 | 0 | 0 | 0 | 0 |
| **6** | 0 | 0 | 0 | 0 | 0 | -1 | 1 | 0 | 0 | 1 |

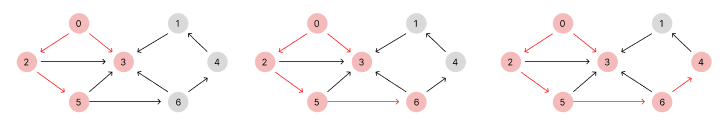
Список смежных вершин:

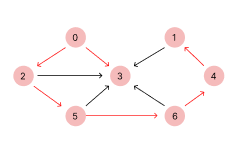
|  |  |
| --- | --- |
| **0** | 2,3 |
| **1** | 3,4 |
| **2** | 0,3,5 |
| **3** | 0,1,2,5,6 |
| **4** | 1,6 |
| **5** | 2,3,6 |
| **6** | 3,4,5 |

***Задание 2.***

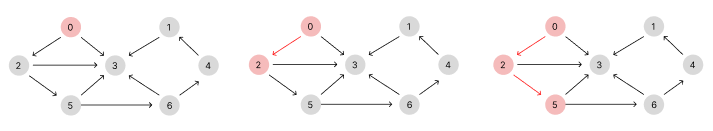
Алгоритм поиска в ширину:

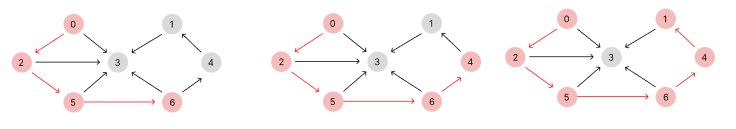


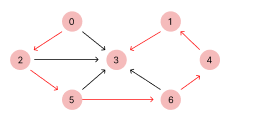




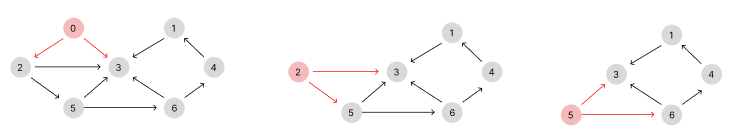
Алгоритм поиска в глубину:

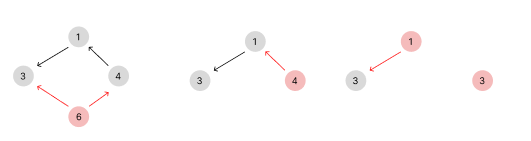


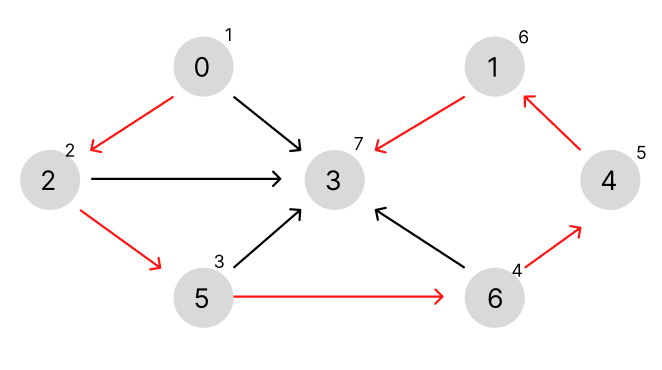




Алгоритма топологической сортировки:







Результат: 0, 2, 5, 6, 4, 1, 3

***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.

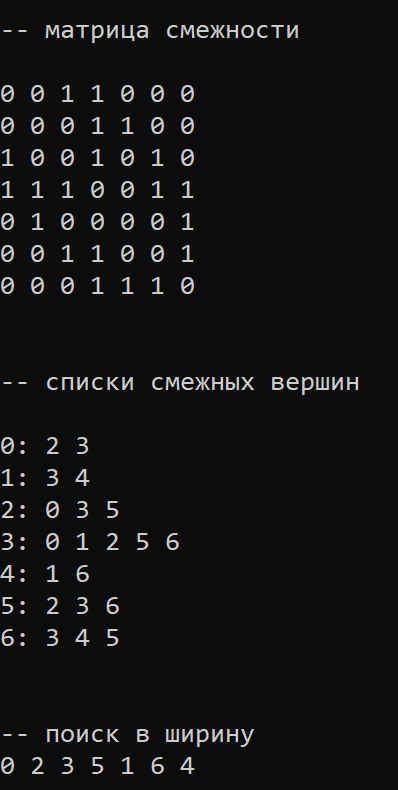
|  |
| --- |
| -- Graph.h  #pragma once  #include <list>  namespace graph  {  struct AList;  struct AMatrix  {  int nV;  int\* mr;  AMatrix(int n, int mr[]);  AMatrix(const AList& al);  void set(int i, int j, int r);  int get(int i, int j)const;  };  struct AList  {  int nV;  std::list<int>\* mr;  void create(int n);  AList(int n, int mr[]);  AList(const AMatrix& am);  void add(int i, int j);  int size(int i) const;  int get(int i, int j)const;  };  }; |

|  |
| --- |
| -- Graph.cpp  #include "stdafx.h"  #include "Graph.h"  namespace graph  {  AMatrix::AMatrix(int n, int mr[])  {  this->nV = n;  this->mr = mr;  };  AMatrix::AMatrix(const AList& al)  {  this->nV = al.nV;  this->mr = new int[this->nV \* this->nV];  for (int k = 0; k < this->nV \* this->nV; k++)  mr[k] = 0;  for (int i = 0; i < this->nV; 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->nV + j] = r;  };  int AMatrix::get(int i, int j)const  {  return this->mr[i \* this->nV + j];  };  //------------List  void AList::create(int n)  {  this->mr = new std::list<int>[this->nV = n];  };  AList::AList(const AMatrix& am)  {  this->create(am.nV);  for (int i = 0; i < this->nV; i++)  for (int j = 0; j < this->nV; j++)  if (am.get(i, j) != 0)  this->add(i, j);  };  AList::AList(int n, int mr[])  {  this->create(n);  for (int i = 0; i < this->nV; i++)  for (int j = 0; j < this->nV; j++)  if (mr[i \* this->nV + 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  {  const static int INF = 0x7fffffff;  const static int NIL = -1;  enum Color { WHITE, GRAY, BLACK };  const graph::AList\* al;  Color\* c;  int\* d;  int\* p;  std::queue<int> q;  BFS(const graph::AList& al, int s);  BFS(const graph::AMatrix& am, int s);  void init(const graph::AList& al, int s);  int get();  }; |

|  |
| --- |
| -- 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->nV];  this->d = new int[this->al->nV];  this->p = new int[this->al->nV];  for (int i = 0; i < this->al->nV; 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;  } |

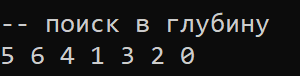
Результат выполнения:



***Задание 4.*** Разработать функцию **DFS**  обхода вершин графа, используя метод поиска глубину.

|  |
| --- |
| -- DFS.h  #pragma once  #include "Graph.h"  #include <vector>  struct DFS  {  const static int NIL = -1;  enum Color { WHITE, GRAY, BLACK };  const graph::AList\* al;  Color\* c;  int\* d;  int\* f;  int\* p;  int t;  DFS(const graph::AList& al);  DFS(const graph::AMatrix& am);  std::vector <int> topological\_sort;  void visit(int v);  void init(const graph::AList& al);  int get(int i);  }; |

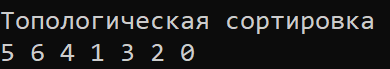
Результат выполнения:



***Задание 5.*** Доработайте функцию **DFS**,для выполнения топологической сортировки графа.

|  |
| --- |
| -- 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->nV];  this->d = new int[this->al->nV];  this->f = new int[this->al->nV];  this->p = new int[this->al->nV];  this->t = 0;  for (int i = 0; i < this->al->nV; i++)  {  this->c[i] = WHITE;  this->d[i] = this->f[i] = 0;  this->p[i] = NIL;  };  for (int i = 0; i < this->al->nV; 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->nV; k++)  if (this->f[k] < min1 && this->f[k] > min2)  {  min1 = this->f[k];  ntx = k;  };  min2 = min1; min1 = INF;  };  return ntx;  }; |

Результат выполнения:



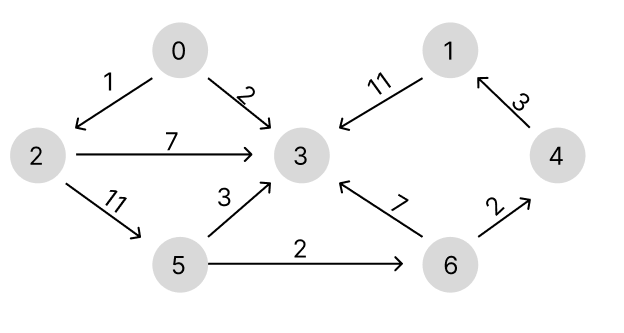
***Задание 6.***

Веса ребер W:

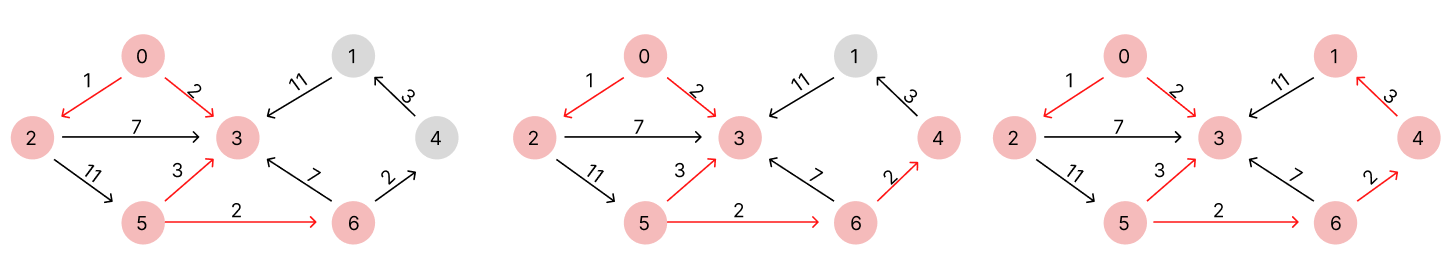
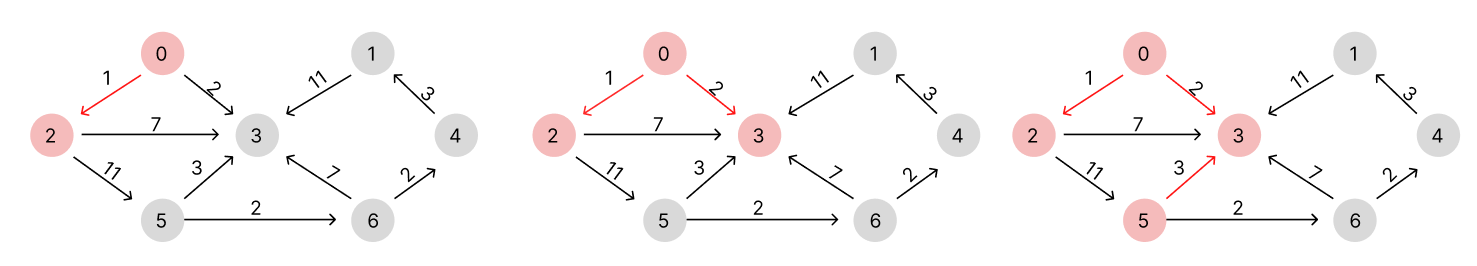
W(e0,1)=8; W(e0,2)=1; W(e0,3)=2; W(e1,3)=11;

W(e4,1)=3;W(e2,3)=7; W(e2,5)=11; W(e6,4)=2;

W(e5,6)=2; W(e5,3)=3; W(e6,3)=7;



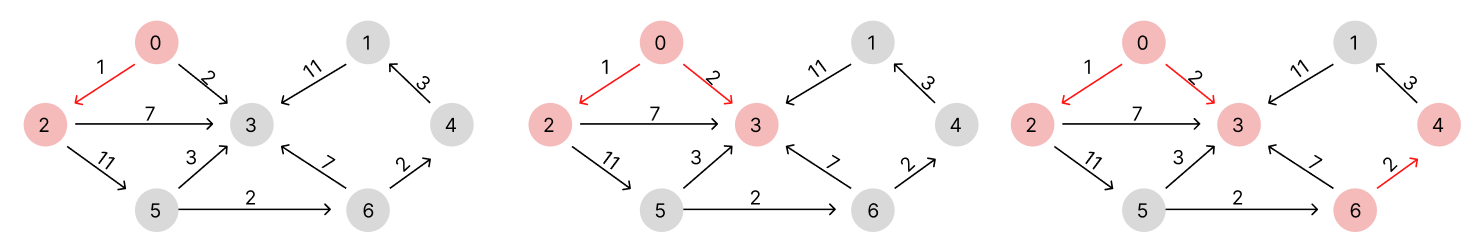
Алгоритм Прима:

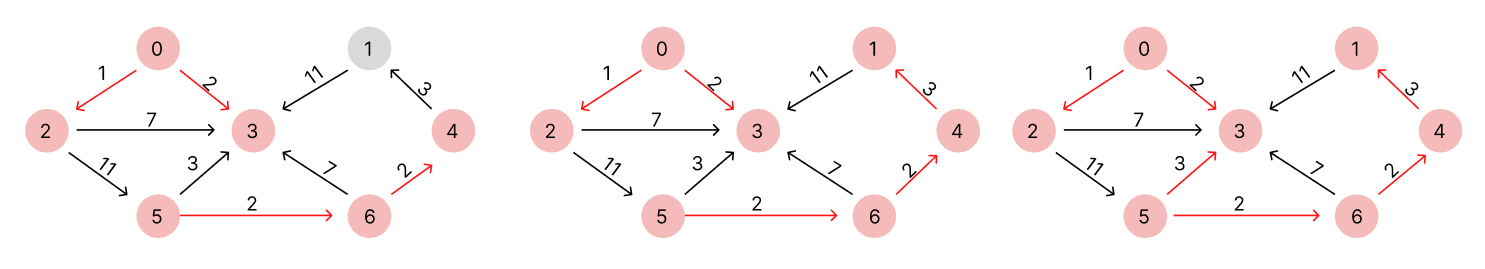


Сумма веса ребер: 13.

***Задание 7.***

Алгоритм Крускала:





Сумма веса ребер: 13