МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»
Факультет: Информационных технологий
Кафедра: Программной инженерии

Отчёт
По дисциплине “Математическое программирование”
На тему “Алгоритмы на графах”

Выполнила: студентка 2 курса 5 группы
специальности ПОИТ Городилина А. С.

Минск
2024

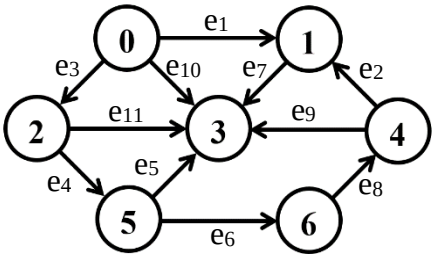
Лабораторная работа 6. Алгоритмы на графах.

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

Ход Работы

1. Представление графа в виде матрицы смежности, матрицы инцидентности, списка смежных вершин.

Вариант 6



Матрица смежности:

	V_0	V_1	V_2	V_3	V_4	V_5	V_6
V_0	0	1	1	1	0	0	0
V_1	0	0	0	1	0	0	0
V_2	0	0	0	1	0	1	0
V_3	0	0	0	0	0	0	0
V_4	0	1	0	1	0	0	0
V_5	0	0	0	1	0	0	1
V_6	0	0	0	0	1	0	0

Матрица инцидентности:

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	E_{11}
V_0	1	0	1	0	0	0	0	0	0	1	0
V_1	-1	-1	0	0	0	0	1	0	0	0	0
V_2	0	0	-1	1	0	0	0	0	0	0	1
V_3	0	0	0	0	-1	0	-1	0	-1	-1	-1
V_4	0	1	0	0	0	0	0	-1	1	0	0
V_5	0	0	0	-1	1	-1	0	0	0	0	0
V_6	0	0	0	0	0	1	0	1	0	0	0

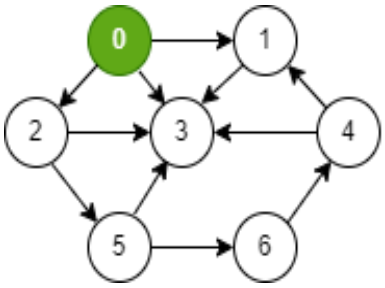
Список смешных вершин:

V_0: V_1, V_2, V_3
V_1: V_3
V_2: V_3, V_5

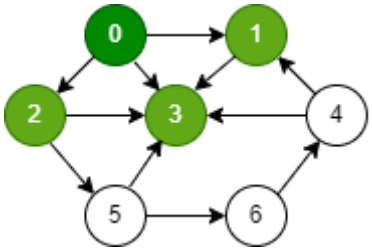
V_3: –
V_4: V_1, V_3
V_5: V_3, V_6
V_6: V_4

2. Алгоритмы поиска в ширину и глубину, топологическая сортировка

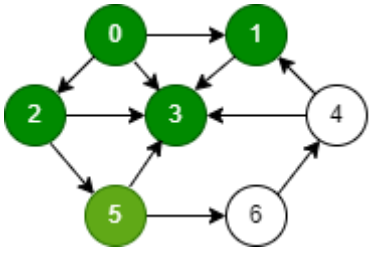
Поиск в ширину (BFS)



Q	**0**						
C	**G**	**W**	**W**	**W**	**W**	**W**	**W**
D	**0**	**I**	**I**	**I**	**I**	**I**	**I**
P	**N**	**N**	**N**	**N**	**N**	**N**	**N**

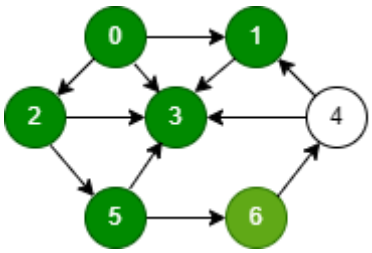


Q	**1**	**2**	**3**				
C	**B**	**G**	**G**	**G**	**W**	**W**	**W**
D	**0**	**1**	**1**	**1**	**I**	**I**	**I**
P	**N**	**0**	**0**	**0**	**N**	**N**	**N**

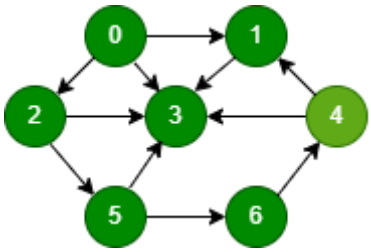


Q	**3**	**5**					
C	**B**	**B**	**B**	**G**	**W**	**G**	**W**
D	**0**	**1**	**1**	**1**	**I**	**2**	**I**
P	**N**	**0**	**0**	**0**	**N**	**2**	**N**

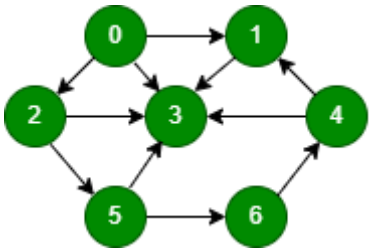
Q	**6**						



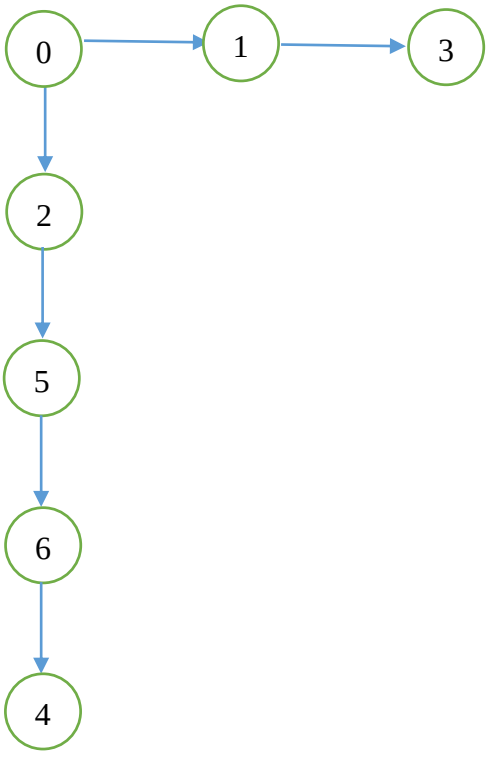
C	B	B	B	B	W	B	G
D	0	1	1	1	I	2	3
P	N	0	0	0	N	2	5



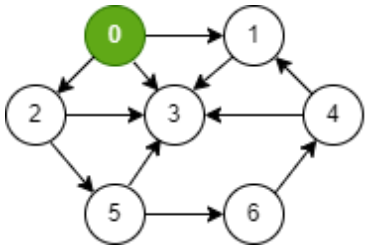
Q	4						
C	B	B	B	B	G	B	B
D	0	1	1	1	4	2	3
P	N	0	0	0	6	2	5



A	0	1	2	3	5	6	4
C	B	B	B	B	B	B	B
D	0	1	1	1	4	2	3
P	N	0	0	0	6	2	5

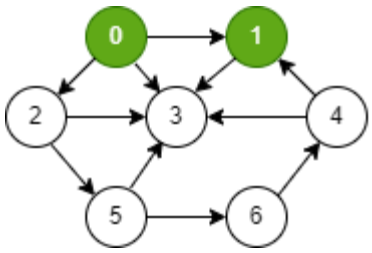


Поиск в глубину (DFS)



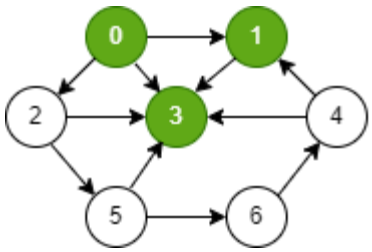
t = 1

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



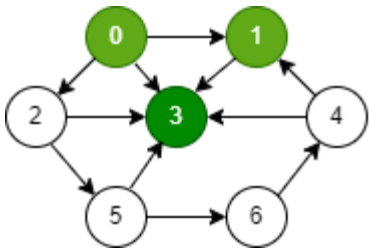
t = 2

C	G	G	W	W	W	W	W
D	1	2	I	I	I	I	I
P	N	0	N	N	N	N	N
F	0	0	0	0	0	0	0



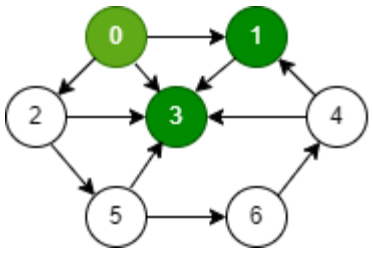
t = 3

C	G	G	W	G	W	W	W
D	1	2	I	3	I	I	I
P	N	0	N	1	N	N	N
F	0	0	0	0	0	0	0



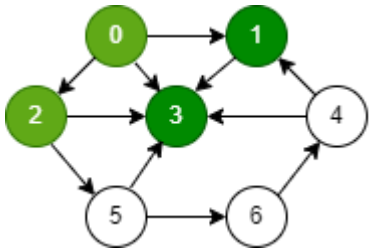
t = 4

C	G	G	W	B	W	W	W
D	1	2	I	3	I	I	I
P	N	0	N	1	N	N	N
F	0	0	0	4	0	0	0



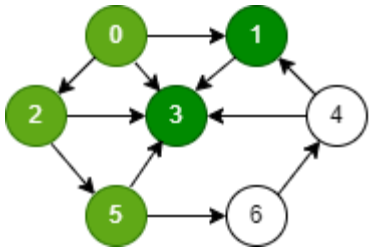
t = 5

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



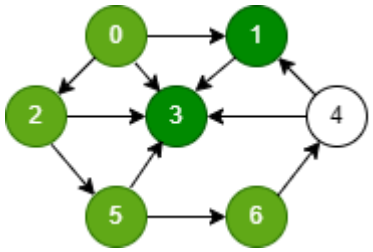
t = 6

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



t = 7

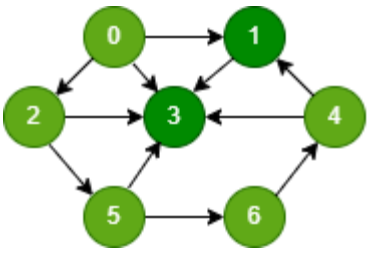
C	G	B	G	B	W	G	W
D	1	2	6	3	I	7	I
P	N	0	0	1	N	2	N
F	0	5	0	4	0	0	0



t = 8

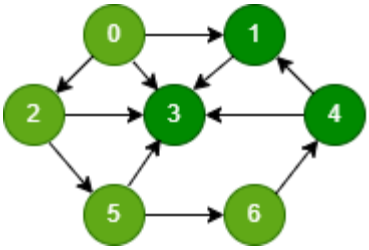
C	G	B	G	B	W	G	G
D	1	2	6	3	I	7	8
P	N	0	0	1	N	2	5
F	0	5	0	4	0	0	0

C	G	B	G	B	G	G	G



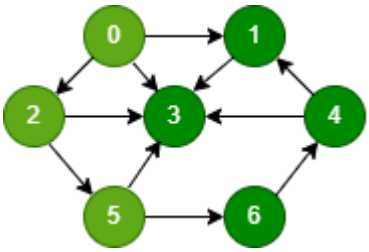
t = 9

D	**1**	**2**	**6**	**3**	**9**	**7**	**8**
P	**N**	**0**	**0**	**1**	**6**	**2**	**5**
F	**0**	**5**	**0**	**4**	**0**	**0**	**0**



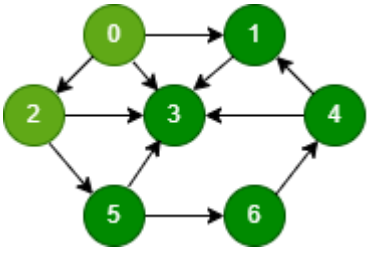
t = 10

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



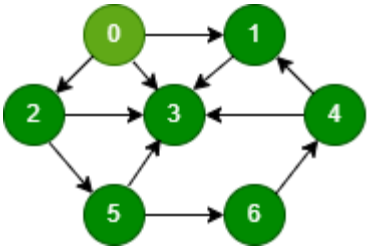
t = 11

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



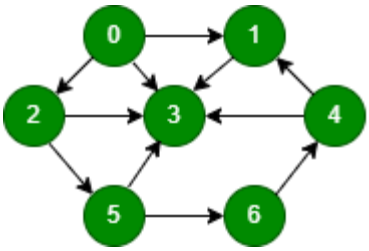
t = 12

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



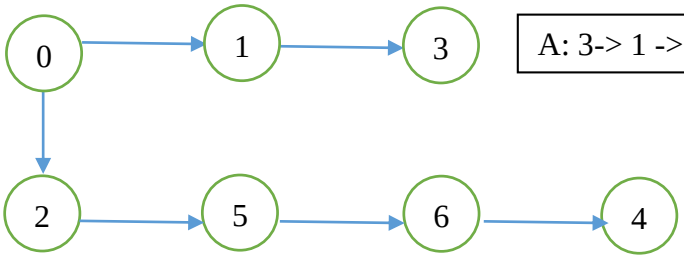
t = 13

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



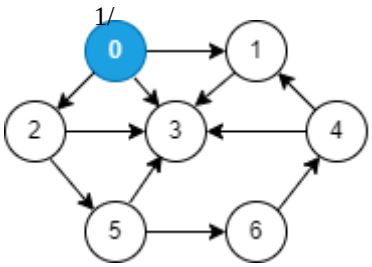
t = 14

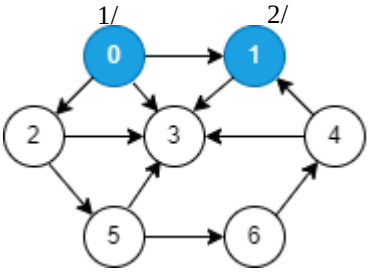
C	B	B	B	B	B	B	B
D	1	2	6	3	9	7	8
P	N	0	0	1	6	2	5
F	14	5	13	4	10	12	11

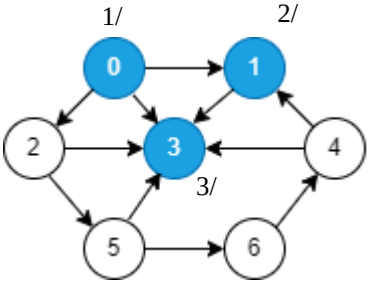


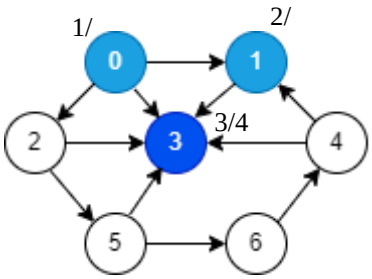
A: 3-> 1 -> 4 -> 6 -> 5 -> 2 -> 0

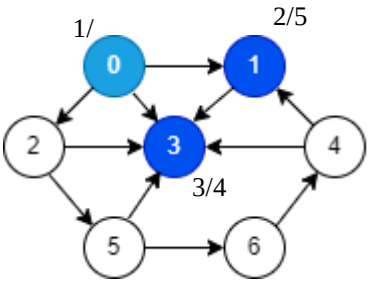
Топологическая сортировка

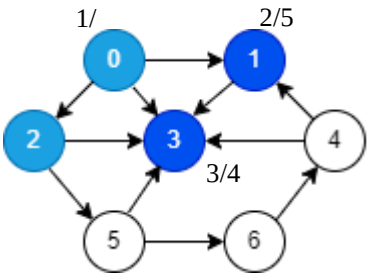








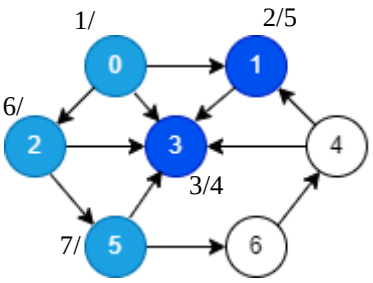




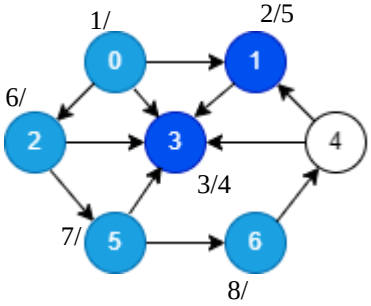
3

1 3

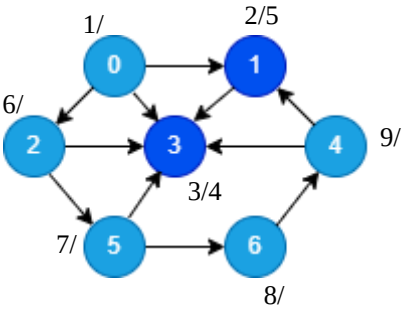
1 3



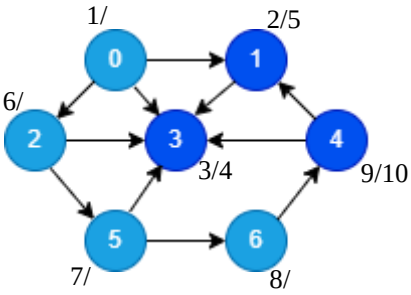
1 3



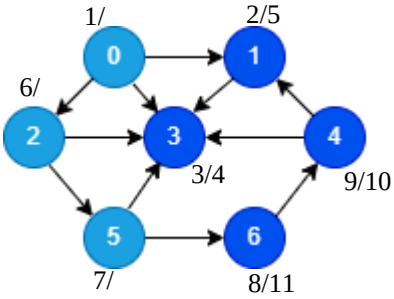
1 3



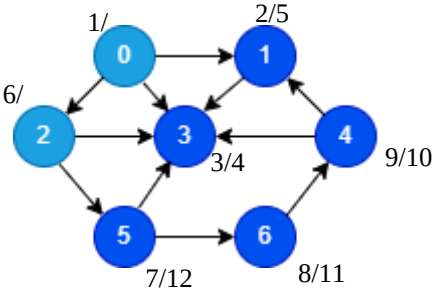
1 3



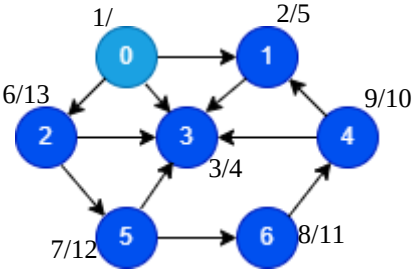
4 1 3



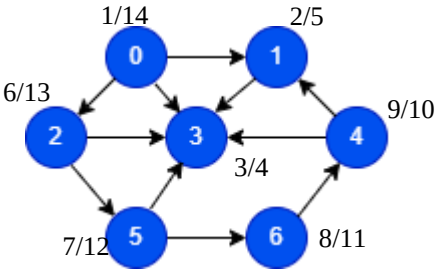
6 4 1 3



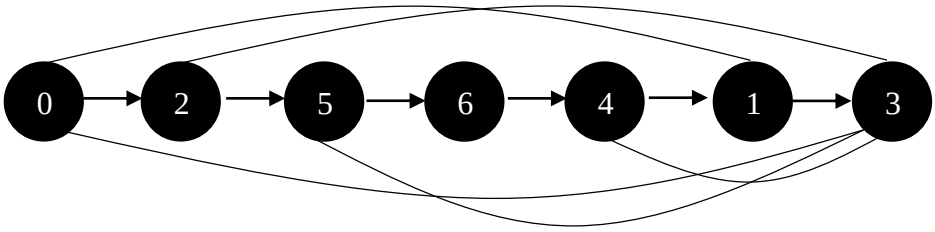
5 6 4 1 3



2 5 6 4 1 3



0 2 5 6 4 1 3



3. Разработка функций для преобразования способов представления, функции для обхода вершин графа (поиск в ширину)

Листинг:

```
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;
};
```

```
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;
}
```

```
-- матрица смежности

0 1 1 1 0 0 0
0 0 0 1 0 0 0
0 0 0 1 0 1 0
0 0 0 0 0 0 0
0 1 0 1 0 0 1
0 0 0 1 0 0 1
0 0 0 0 1 0 0
```

```
-- списки смежных вершин

0: 1 2 3
1: 3
2: 3 5
3:
4: 1 3 6
5: 3 6
6: 4
```

```
-- поиск в ширину
0 1 2 3 5 6 4
```

4-5. Разработать функцию DFS для обхода в глубину и топологической сортировки

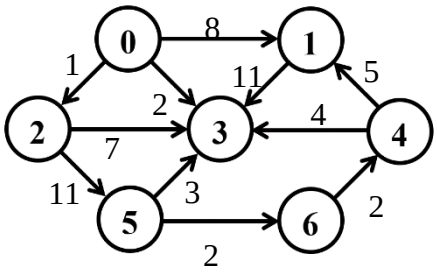
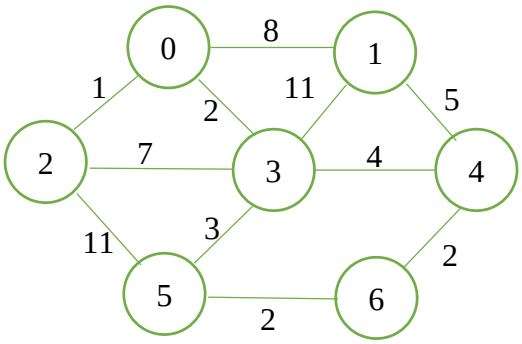
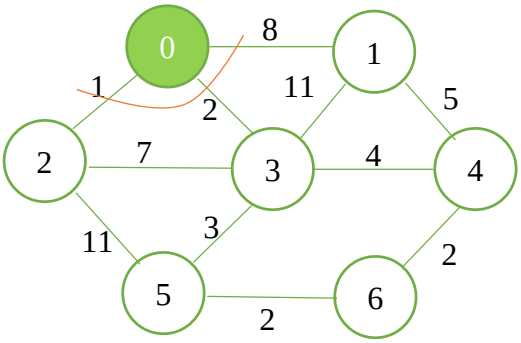
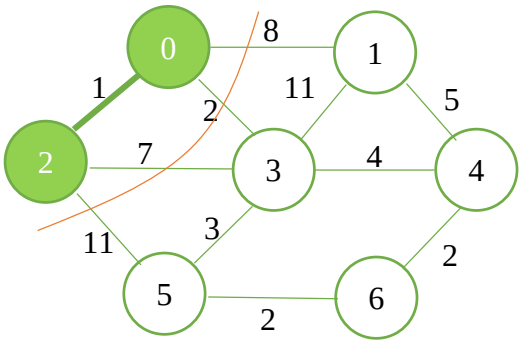
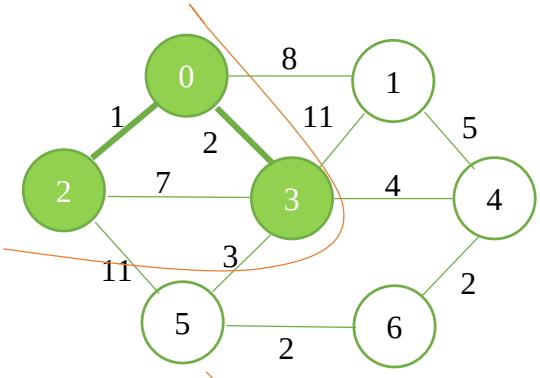
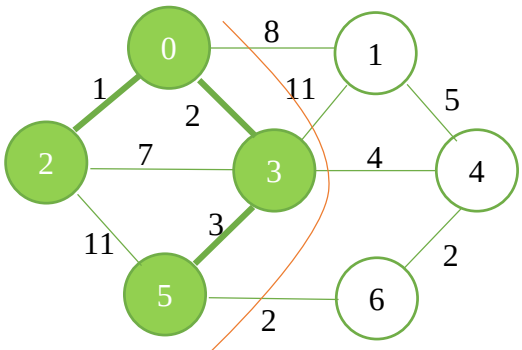
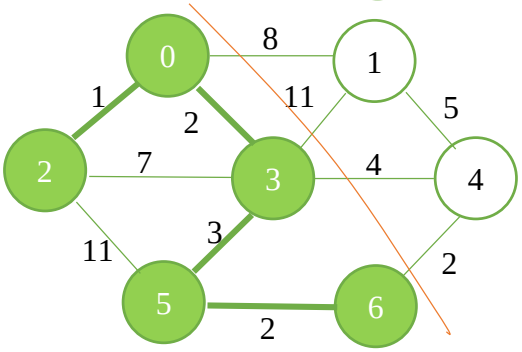
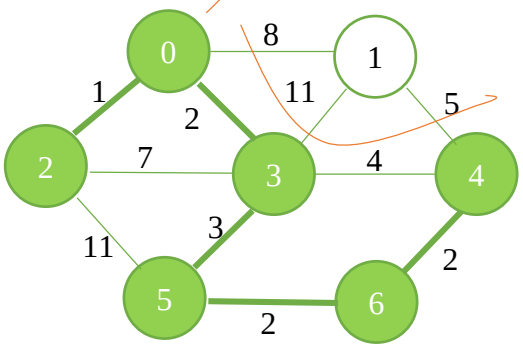
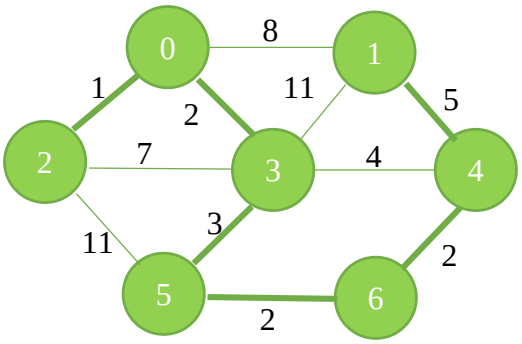
Листинг:

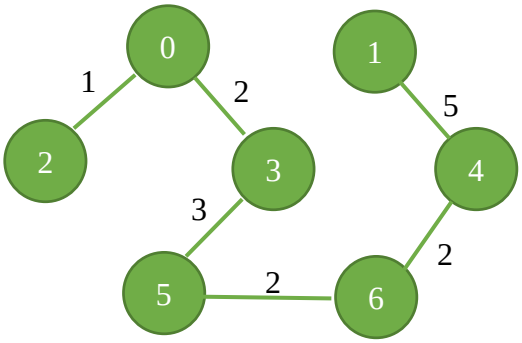
```
void DFS::init(const graph::AList& al)
{
	this->al = &al;
	this->c = new Color[this->al->n_vertex];
	this->d = new int[this->al->n_vertex];
	this->f = new int[this->al->n_vertex];
	this->p = new int[this->al->n_vertex];
	this->t = 0;
	for (int i = 0; i < this->al->n_vertex; i++)
	{
		this->c[i] = 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;
};
```

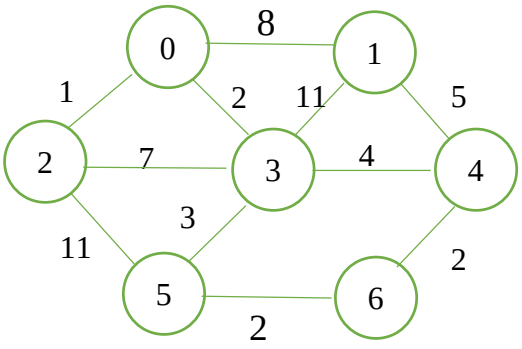
```
-- поиск в глубину
3 1 4 6 5 2 0


-- поиск в глубину (ориентированный граф)
3 1 4 6 5 2 0

Топологическая сортировка
3 1 4 6 5 2 0
```

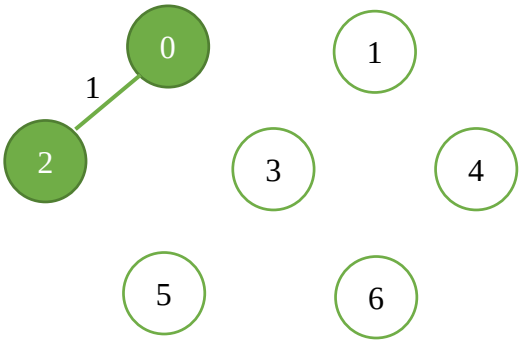
6. Алгоритм Прима

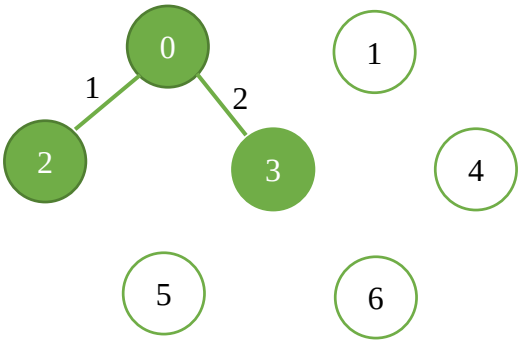










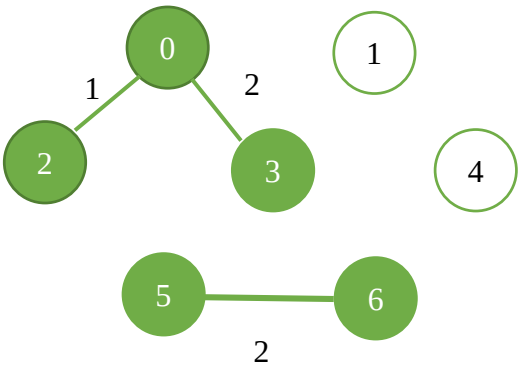


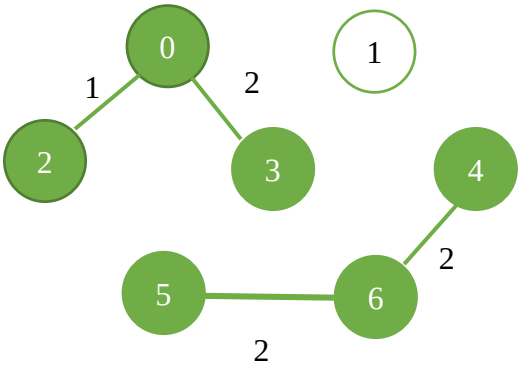
7. Алгоритм Крускала

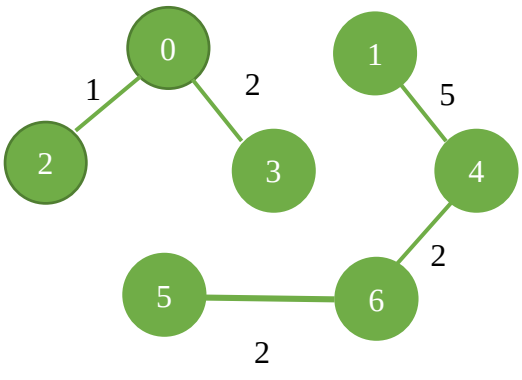












Вывод: в результате выполнения лабораторной работы были изучены основные алгоритмы на графах: поиск в ширину, поиск в глубину, топологическая сортировка, алгоритм Прима и алгоритм Крускала.