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

|  |  |
| --- | --- |
| Вариант | Граф G |
| 2 |  |

Матрица смежности - это квадратная матрица размера n x n, где n - число вершин в графе. Эта матрица используется для представления графа в виде таблицы, где каждый элемент матрицы a\_ij показывает, существует ли ребро между вершинами i и j.

**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** |

Изображение выглядит как круг, рисунок, графическая вставка

Автоматически созданное описание

**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** |

**Список смежных вершин:**

S0 = {2}

S1 = ∅

S2 = {3, 5}

S3 = {1, 4, 5, 6}

S4 = {1, 6}

S5 = {6}

S6 = ∅

***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.

**Алгоритм поиска в ширину (англ. breadth-first search, BFS)** позволяет найти кратчайшие пути из одной вершины невзвешенного графа до всех остальных вершин

**Поиск в ширину:**

Q- для промежуточного хранения вершин(очередь)



Шаг 1

Очередь : 0

Последовательность: 0



Шаг 2

Очередь : 2

Последовательность: 0, 2



Шаг 3

Очередь : 3,5

Последовательность: 0, 2, 3



Шаг 4

Очередь : 5,1,4,6

Последовательность: 0, 2, 3, 5



Шаг 5

Очередь : 1,4,6

Последовательность: 0, 2, 3, 5, 1



Шаг 6

Очередь : 4,6

Последовательность: 0, 2, 3, 5, 1, 4



Шаг 7

Очередь : 6

Последовательность: 0, 2, 3, 5, 1, 4, 6



Шаг 8

Очередь :





Обход в ширину: 0,2,3,5,1,4,6

**Алгоритм поиска (или обхода) в глубину** (англ. depth-first search, DFS) позволяет построить обход графа, при котором посещаются все вершины, доступные из начальной вершины.

**Алгоритм поиска в глубину:**

T-шаг



Шаг 1

Текущая вершина: 0

Стек: 0, 2

Последовательность: 0



Шаг 2

Текущая вершина: 2

Стек: 0, 2, 3

Последовательность: 0, 2



Шаг 3

Текущая вершина: 3

Стек: 0, 2, 3, 6, 4,1

Последовательность: 0, 2, 3



Шаг 4

Текущая вершина: 1

Стек: 0, 2, 3, 6, 4,1

Последовательность: 0, 2, 3, 1



Шаг 5

Текущая вершина: 4

Стек: 0, 2, 3, 6, 4

Последовательность: 0, 2, 3, 1, 4



Шаг 6

Текущая вершина: 6

Стек: 0, 2, 3,4,6

Последовательность: 0, 2, 3, 1, 4, 6



Шаг 7

Текущая вершина: 4

Стек: 0, 2, 3,4

Последовательность: 0, 2, 3, 1, 4, 6



Шаг 8

Текущая вершина: 3

Стек: 0, 2, 3, 5

Последовательность: 0, 2, 3, 1, 4, 6



Шаг 9

Стек: 0, 2, 3

Последовательность: 0, 2, 3, 1, 4, 6, 5



Шаг 12

Стек: 0, 2

Последовательность: 0, 2, 3, 1, 4, 6, 5



Шаг 13

Стек: 0

Последовательность: 0, 2, 3, 1, 4, 6, 5



Шаг 14

Стек:





Обход в глубину: 0, 2, 3, 1, 4, 6, 5

**Топологическая сортировка**

**Топологическая сортировка −** это процедура упорядочивания вершин бесконтурного ориентированного графа, не имеющего циклов (ациклического графа). 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**.**













Последовательность: 6

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **6** |  |  |  |  |  |  |



Последовательность: 6, 5

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









Последовательность: 6, 5, 1

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



Последовательность: 6, 5, 1, 4

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



Последовательность: 6, 5, 1, 4, 3

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



Последовательность: 6, 5, 1, 4, 3, 2

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

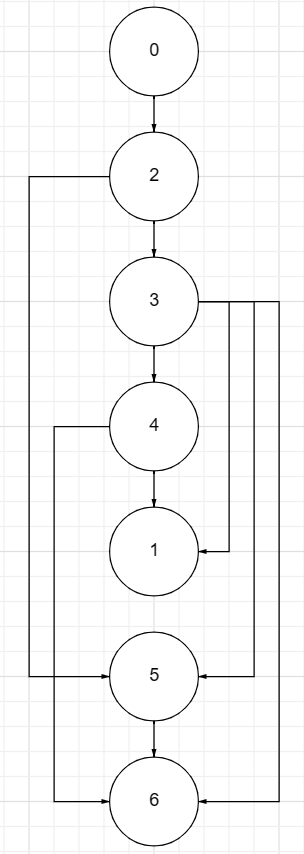


Последовательность: 6, 5, 1, 4, 3, 2, 0

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

**Результат**

Топологическая сортировка: 6 5 1 4 3 2 0



***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.

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

Листинг 3.1 — файл BFS.h

|  |
| --- |
| // ---BFS.cpp  //  #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 3.2 — файл BFS.cpp

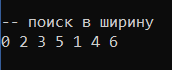


Рисунок 1 – Демонстрация программы с функцией BFS

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

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

Листинг 4.1 — файл DFS.h

|  |
| --- |
| // ---DFS.cpp  //  #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++) // 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;  }; |

Листинг 4.2 — файл DFS.cpp

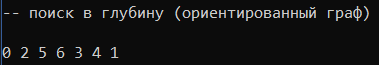


Рисунок 2 – Демонстрация работы функции DFS

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

|  |
| --- |
| // Функция для топологической сортировки  void topologicalSort(int v, bool visited[], std::stack<int>& Stack, graph::AList& g)  {  // Помечаем текущую вершину как посещенную  visited[v] = true;  // Рекурсивно вызываем функцию для всех смежных вершин  // Если смежная вершина не была посещена, то рекурсивно вызываем функцию  for (int i = 0; i < g.size(v); i++)  if (!visited[g.get(v, i)])  topologicalSort(g.get(v, i), visited, Stack, g);  // Помещаем текущую вершину в стек  Stack.push(v);  }  . . .  std::cout << std::endl << "-- топологическая сортировка" << std::endl;  // Создаем стек для хранения топологической сортировки  std::stack<int> Stack;  // Массив для хранения информации о посещении вершин  bool \*visited = new bool[g5.n\_vertex];  for (int i = 0; i < g5.n\_vertex; i++) visited[i] = false;  // Вызываем функцию для топологической сортировки  for (int i = 0; i < g5.n\_vertex; i++)  if (visited[i] == false)  topologicalSort(i, visited, Stack, g5);  // Выводим топологически отсортированный граф  while (Stack.empty() == false)  {  std::cout << Stack.top() << " ";  Stack.pop();  }  std::cout << std::endl; |

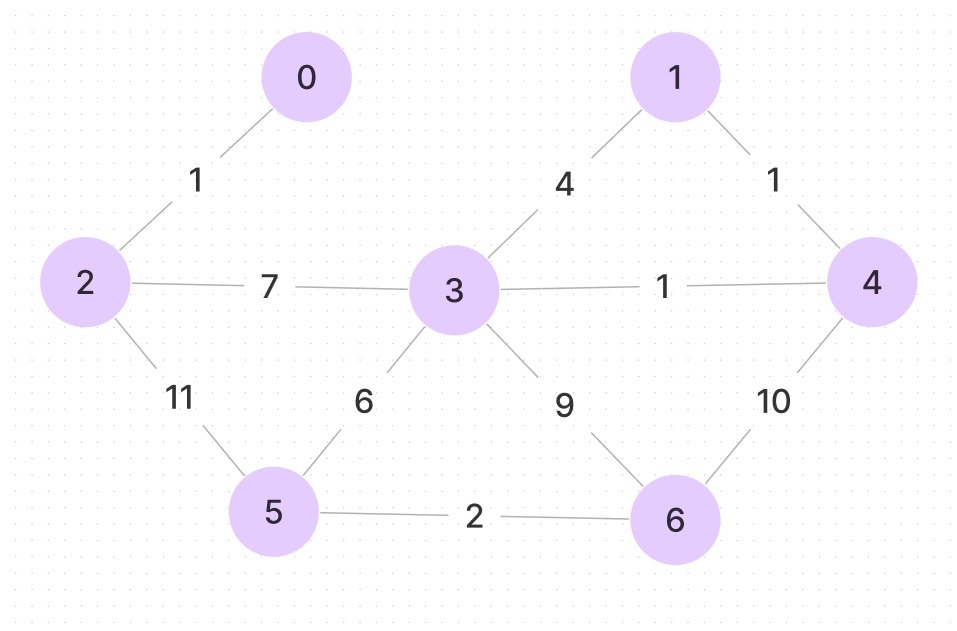
Листинг 5.1— функция топологической сортировки



Рисунок 3 – Демонстрация работы функции DFS

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

Веса ребер принять:

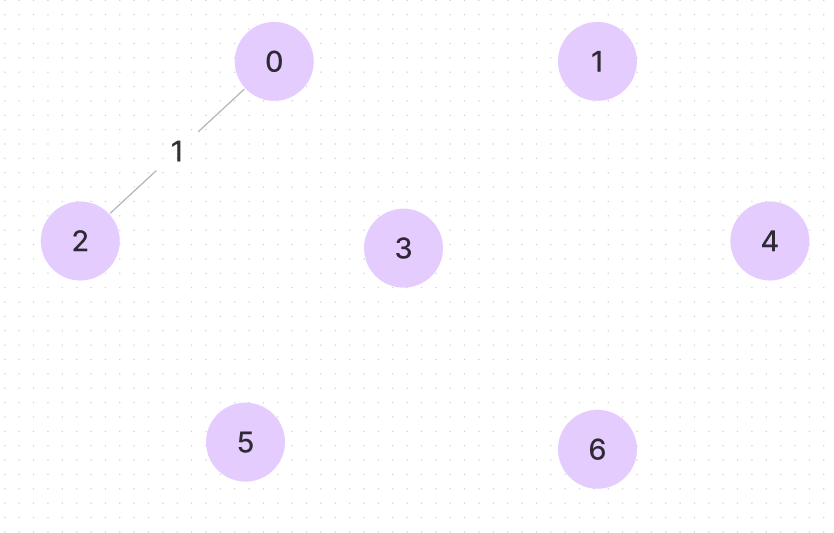


Шаг 1:

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}

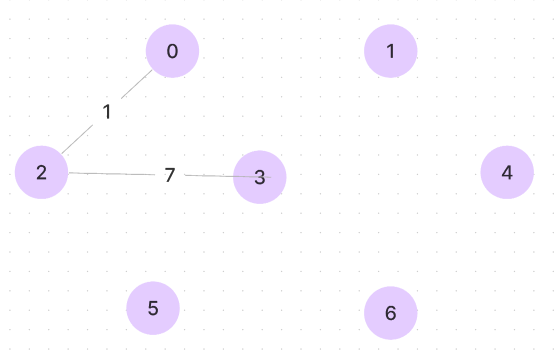
Вершина 0 соединена с вершинами 2. Вес ребра равен 1. Тогда, вершину 2 включаем в множество U, а ребро 0-2 — в множество T.



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.

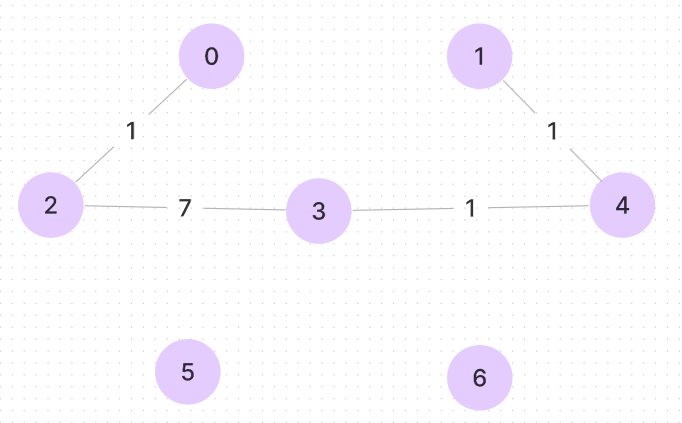
Изображение выглядит как круг, снимок экрана, фиолетовый

Автоматически созданное описание

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.

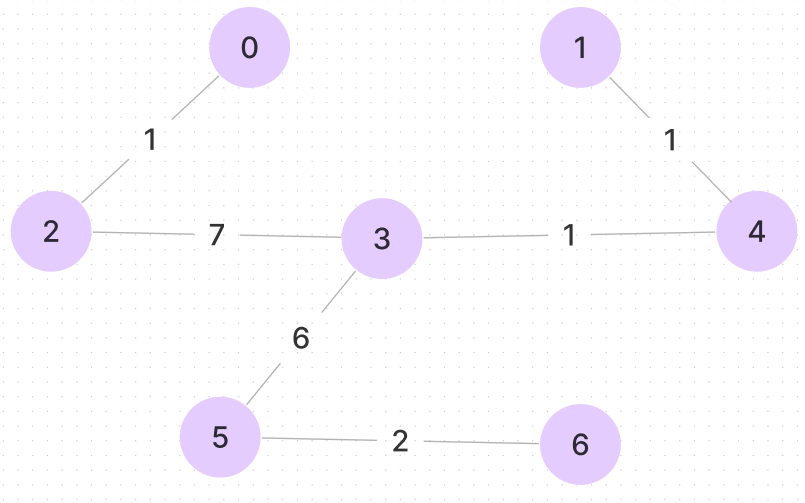
Изображение выглядит как круг

Автоматически созданное описание

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

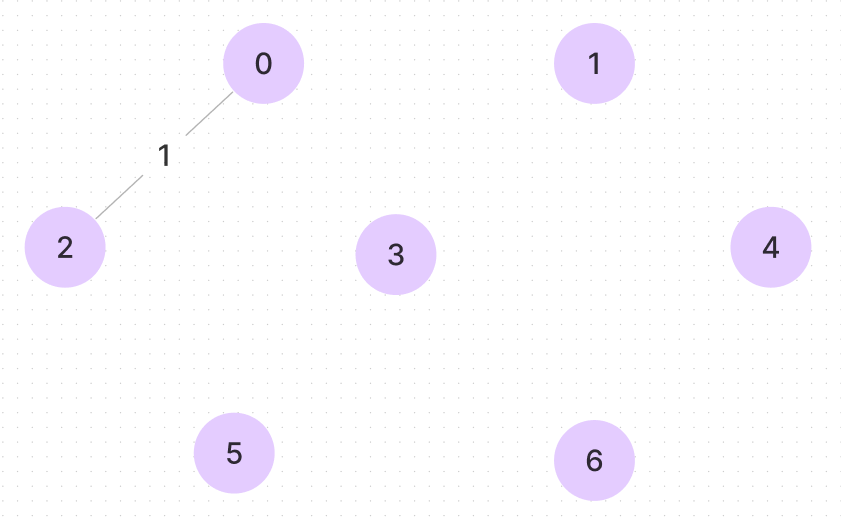
***Задание 7.*** По графу, соответствующему варианту составить минимальное остовное дерево по алгоритму Крускала. Шаги построения отразить в отчете

Изображение выглядит как круг, диаграмма, линия

Автоматически созданное описание

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.



Шаг 2:

Изображение выглядит как круг, снимок экрана

Автоматически созданное описание

Шаг 3:

Изображение выглядит как круг

Автоматически созданное описание

Шаг 4:

Изображение выглядит как круг

Автоматически созданное описание

Шаг 5:

Изображение выглядит как круг

Автоматически созданное описание

Шаг 6:

Изображение выглядит как круг, снимок экрана, диаграмма

Автоматически созданное описание

Вес минимального остовного дерева: 18

Вывод: в ходе лабораторной работы были освоены сущность и программная реализация: а) способов представления графов; б) алгоритмов поиска в ширину и глубину; в) алгоритма топологической сортировки графов и разобрала алгоритм Прима и алгоритм Крускала.