ГУАП

# КАФЕДРА №43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  | 23.11.2023 |  | С.А. Рогачев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 7 |
| АЛГОРИТМЫ НА ГРАФАХ |
| по курсу: АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4232 |  | 23.11.2023 |  | М.В.Куриш |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

1. **Цель работы**

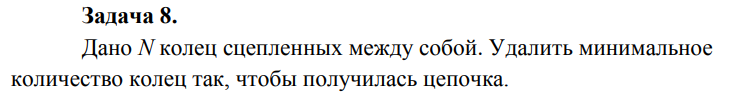
Целью работы является изучение графов и получение практических навыков их использования.

1. **Задание на лабораторную работу**

Разработать на языке программирования высокого уровня программу, которая должна выполнять функцию, в соответствии с вариантом задания. Варианты задания приведены в таблице 7 (формулировки задач приведены после таблицы).

Вариант 22.

|  |  |  |
| --- | --- | --- |
| **№ вар.** | **Задача** | **Представление графа** |
| 22 | 8 | Матрица смежности |



1. **Листинг программы**

|  |
| --- |
| #include <iostream>  #include <random>  #include <string>  using namespace std;  struct Pair {  public:  int val;  int prev;  Pair(int val, int prev) {  this->val = val;  this->prev = prev;  }  Pair() {}  };  class Queue {  private:  class QueueNode {  public:  Pair val;  QueueNode\* next;  QueueNode(int val, int prev) {  this->val = Pair(val, prev);  this->next = nullptr;  }  };  QueueNode\* head;  QueueNode\* tail;  public:  Queue() {  head = nullptr;  tail = nullptr;  }  Pair peek()  {  if (!isEmpty()) {  return tail->val;  }  throw runtime\_error("Попытка чтения из пустой очереди");  }  Pair pop()  {  if (!isEmpty()) {  QueueNode\* tempNode = tail;  tail = tail->next;  Pair tempVal = tempNode->val;  delete tempNode;  if (tail == nullptr)  head = nullptr;  return tempVal;  }  throw runtime\_error("Попытка чтения из пустой очереди");  }  void push(int val, int prev)  {  if (isEmpty()) {  head = new QueueNode(val, prev);  tail = head;  }  else {  head->next = new QueueNode(val, prev);  head = head->next;  }  }  void clear()  {  while (!isEmpty()) {  pop();  }  }  bool isEmpty()  {  return tail == nullptr;  }  };  int intInput(string msg);  void autoFillGraph(int\*\* graph, int size);  void fillGraph(int\*\* graph, int size);  void testCase();  void showGraph(int\*\* graph, int size);  void chainPassage(int\*\* graph, int size, int parent, bool\* checked);  void showChain(int\*\* graph, int size);  int\*\* shrinkGraph(int\*\* graph, int\* size);  int\*\* graphCopy(int\*\* graph, int size);  bool graphsEquals(int\*\* graph1, int\*\* graph2, int size);  bool arrContains(int\* arr, int size, int val);  int\* addAsLast(int\* arr, int size, int val);  int lengthOfPath(int\* path, int size);  int\* DFS\_longestPath(int\*\* graph, int size, int\* path, int last);  int\*\* BFS\_deletingCycles(int\*\* graph, int size, int first);  void excludeRing(int\*\* graph, int size, int ring);  int\*\* deleteRing(int\*\* graph, int size, int ring);  int\*\* removeRings(int\*\* graph, int size);  int main()  {  setlocale(LC\_ALL, "RUS");  int g\_size = 0;  int\*\* graph = nullptr;  bool graph\_is\_created = false;  bool flag1 = true;  bool flag2;  while (flag1) {  cout << "\n~~~~~~~~~~~~ Граф и кольца ~~~~~~~~~~~~\n";  cout << "1 - Создать граф\n";  cout << "2 - Показать граф\n";  cout << "3 - Удалить ненужные кольца\n";  cout << "4 - Показать цепочку\n";  cout << "5 - Очистить граф от несцепленных колец\n";  cout << "-------------------\n";  cout << "6 - Тестовый набор\n";  cout << "-------------------\n";  cout << "10 - Очистка экрана\n";  cout << "100 - Завершение работы\n";  switch (intInput("Выбор: ")) {  case 1:  g\_size = intInput("Введите размер графа: ");  graph = new int\* [g\_size];  for (int i = 0; i < g\_size; i++)  graph[i] = new int[g\_size];  flag2 = true;  while (flag2) {  cout << "-----Создание графа-----\n";  cout << "1 - Вручную\n";  cout << "2 - Автоматически\n";  switch (intInput("Выбор: ")) {  case 1:  fillGraph(graph, g\_size);  flag2 = false;  break;  case 2:  autoFillGraph(graph, g\_size);  flag2 = false;  break;  default:  cout << "\x1B[31mНеверный выбор пункта меню\033[0m\nПовторите попытку\n";  break;  }  }  showGraph(graph, g\_size);  graph\_is\_created = true;  break;  case 2:  if (graph\_is\_created) {  showGraph(graph, g\_size);  }  else  cout << "\x1B[33mДля начала создайте граф\033[0m\n";  break;  case 3:  if (graph\_is\_created) {  graph = removeRings(graph, g\_size);  showGraph(graph, g\_size);  }  else  cout << "\x1B[33mДля начала создайте граф\033[0m\n";  break;  case 4:  if (graph\_is\_created) {  showChain(graph, g\_size);  }  else  cout << "\x1B[33mДля начала создайте граф\033[0m\n";  break;  case 5:  if (graph\_is\_created) {  graph = shrinkGraph(graph, &g\_size);  showGraph(graph, g\_size);  }  else  cout << "\x1B[33mДля начала создайте граф\033[0m\n";  break;  case 6:  testCase();  break;  case 10:  system("cls");  break;  case 100:  flag1 = false;  break;  default:  cout << "\x1B[31mНеверный выбор пункта меню\033[0m\nПовторите попытку\n";  break;  }  }  }  int intInput(string welcome) {  int a;  cout << welcome;  cin >> a;  while (!cin) {  cout << "\x1B[31mНеверный ввод. (Ожидается число)\033[0m\n";  cin.clear();  cin.ignore(999, '\n');  cout << "Введите заново: ";  cin >> a;  }  return a;  }  void autoFillGraph(int\*\* graph, int size)  {  random\_device rd;  mt19937 rng(rd());  uniform\_int\_distribution<int> rings(0, 1);  for (int i = 0; i < size; i++) {  graph[i][i] = 0;  bool rowCorrect = i == size - 1;  while (!rowCorrect) {  for (int j = i + 1; j < size; j++) {  graph[i][j] = rings(rng);  graph[j][i] = graph[i][j];  if (graph[i][j] == 1)  rowCorrect = true;  }  }  }  }  void fillGraph(int\*\* graph, int size)  {  for (int i = 0; i < size; i++) {  graph[i][i] = 0;  bool rowCorrect = i == size - 1;  while (!rowCorrect) {  for (int j = i + 1; j < size; j++) {  graph[i][j] = intInput("(1 - если кольцо #" + to\_string(i) + " связано с кольцом #" + to\_string(j) + ") Ввод: ");  graph[j][i] = graph[i][j];  if (graph[i][j] == 1)  rowCorrect = true;  }  if (!rowCorrect)  cout << "\x1B[31mГраф не может иметь несцепленные кольца!\033[0m\n";  }  }  }  void testCase()  {  cout << "\_\_Граф #1\_\_\n";  int g\_size = 5;  int\*\* graph = new int\* [g\_size];  for (int i = 0; i < g\_size; i++)  graph[i] = new int[g\_size];  for (int i = 0; i < g\_size; i++) {  graph[i][i] = 0;  for (int j = i + 1; j < g\_size; j++) {  if (i == 0 && j == 2 ||  i == 1 && j == 3 ||  i == 2 && j == 4 ||  i == 3 && j == 1 ||  i == 3 && j == 4)  graph[i][j] = 1;  else  graph[i][j] = 0;  graph[j][i] = graph[i][j];  }  }  showGraph(graph, g\_size);  showChain(graph, g\_size);  //////////////////////////////////////////////  cout << "\_\_Граф #2\_\_\n";  for (int i = 0; i < g\_size; i++) {  graph[i][i] = 0;  for (int j = i + 1; j < g\_size; j++) {  if (i == 0 && j == 1 ||  i == 0 && j == 3 ||  i == 1 && j == 2 ||  i == 2 && j == 3 ||  i == 2 && j == 4 ||  i == 3 && j == 4)  graph[i][j] = 1;  else  graph[i][j] = 0;  graph[j][i] = graph[i][j];  }  }  showGraph(graph, g\_size);  showChain(graph, g\_size);  cout << "Удаление колец\n";  graph = removeRings(graph, g\_size);  graph = shrinkGraph(graph, &g\_size);  showGraph(graph, g\_size);  showChain(graph, g\_size);  //////////////////////////////////////////////  cout << "\_\_Граф #3\_\_\n";  for (int i = 0; i < g\_size; i++)  delete[] graph[i];  delete[] graph;  g\_size = 7;  graph = new int\* [g\_size];  for (int i = 0; i < g\_size; i++)  graph[i] = new int[g\_size];  for (int i = 0; i < g\_size; i++) {  graph[i][i] = 0;  for (int j = i + 1; j < g\_size; j++) {  if (i == 0 && j == 1 ||  i == 1 && j == 2 ||  i == 2 && j == 3 ||  i == 3 && j == 4 ||  i == 3 && j == 5 ||  i == 4 && j == 6 ||  i == 5 && j == 6 ||  i == 0 && j == 6)  graph[i][j] = 1;  else  graph[i][j] = 0;  graph[j][i] = graph[i][j];  }  }  showGraph(graph, g\_size);  showChain(graph, g\_size);  cout << "Удаление колец\n";  graph = removeRings(graph, g\_size);  showGraph(graph, g\_size);  showChain(graph, g\_size);  //////////////////////////////////////////////  cout << "\_\_Граф #4\_\_\n";  for (int i = 0; i < g\_size; i++)  delete[] graph[i];  delete[] graph;  g\_size = 11;  graph = new int\* [g\_size];  for (int i = 0; i < g\_size; i++)  graph[i] = new int[g\_size];  for (int i = 0; i < g\_size; i++) {  graph[i][i] = 0;  for (int j = i + 1; j < g\_size; j++) {  if (i == 0 && j == 8 ||  i == 1 && j == 2 ||  i == 1 && j == 8 ||  i == 3 && j == 4 ||  i == 3 && j == 8 ||  i == 3 && j == 9 ||  i == 3 && j == 10 ||  i == 4 && j == 5 ||  i == 5 && j == 6 ||  i == 5 && j == 7 ||  i == 6 && j == 7 ||  i == 6 && j == 10 ||  i == 9 && j == 10)  graph[i][j] = 1;  else  graph[i][j] = 0;  graph[j][i] = graph[i][j];  }  }  showGraph(graph, g\_size);  showChain(graph, g\_size);  removeRings(graph, g\_size);  showGraph(graph, g\_size);  showChain(graph, g\_size);  graph = shrinkGraph(graph, &g\_size);  showGraph(graph, g\_size);  }  void showGraph(int\*\* graph, int size)  {  cout << "Граф:\n ";  cout << "\x1B[4m\x1B[94m";  for (int i = 0; i < size; i++) {  cout << i << " ";  }  cout << "\033[24m\033[0m";  cout << endl;  for (int i = 0; i < size; i++) {  cout << "\x1B[94m" << i << "|\033[0m ";  for (int j = 0; j < size; j++) {  if (i == j)  cout << "\x1B[31m";  cout << graph[i][j] << " ";  if (i == j)  cout << "\033[0m";  }  cout << endl;  }  cout << endl;  }  void chainPassage(int\*\* graph, int size, int parent, bool\* checked) {  for (int i = 0; i < size; i++) {  if (i == parent || (checked[i] == true))  continue;  if (graph[parent][i] != 0) {  cout << i << " ";  checked[i] = true;  chainPassage(graph, size, i, checked);  break;  }  }  }  void showChain(int\*\* graph, int size)  {  int start = -1;  for (int i = 0; i < size; i++) {  int countLinked = 0;  for (int j = 0; j < size; j++) {  if (graph[i][j] != 0)  countLinked++;  }  if (countLinked == 1) {  start = i;  } else if (countLinked > 2) {  cout << "Граф не представляет собой цепочку\n";  return;  }  }  if (start == -1) {  cout << "Граф не представляет собой цепочку\n";  return;  }  cout << "Цепочка: ";  cout << start << " ";  bool\* checked = new bool[size];  checked[start] = true;  chainPassage(graph, size, start, checked);  cout << endl;  }  int\*\* shrinkGraph(int\*\* graph, int\* size)  {  for (int i = 0; i < \*size; i++) {  bool allNull = true;  for (int j = 0; j < \*size; j++) {  if (graph[i][j] != 0) {  allNull = false;  break;  }  }  if (allNull) {  graph = deleteRing(graph, \*size, i);  (\*size)--;  i = -1;  }  }  return graph;  }  int\*\* graphCopy(int\*\* graph, int size)  {  int\*\* temp = new int\* [size];  for (int i = 0; i < size; i++)  temp[i] = new int[size];  for (int i = 0; i < size; i++) {  for (int j = 0; j < size; j++)  temp[i][j] = graph[i][j];  }  return temp;  }  bool graphsEquals(int\*\* graph1, int\*\* graph2, int size)  {  for (int i = 0; i < size; i++) {  for (int j = 0; j < size; j++)  if (graph1[i][j] != graph2[i][j])  return false;  }  return true;  }  void excludeRing(int\*\* graph, int size, int ring)  {  for (int i = 0; i < size; i++) {  graph[ring][i] = 0;  graph[i][ring] = 0;  }  }  int\*\* deleteRing(int\*\* graph, int size, int ring) {  int\*\* newGraph = new int\* [size - 1];  for (int i = 0; i < size - 1; i++)  newGraph[i] = new int[size - 1];  for (int i = 0, ii = 0; i < size; i++, ii++) {  if (i == ring) {  ii--;  continue;  }  for (int j = 0, jj = 0; j < size; j++, jj++) {  if (j == ring) {  jj--;  continue;  }  newGraph[ii][jj] = graph[i][j];  }  }  for (int i = 0; i < size; i++)  delete[] graph[i];  delete[] graph;  return newGraph;  }  bool arrContains(int\* arr, int size, int val) {  for (int i = 0; i < size; i++)  if (arr[i] == val)  return true;  return false;  }  int\* addAsLast(int\* arr, int size, int val) {  int\* newArr = new int[size];  bool inserted = false;  for (int i = 0; i < size; i++) {  newArr[i] = arr[i];  if (!inserted) {  if (newArr[i] == -1) {  newArr[i] = val;  inserted = true;  }  }  }  return newArr;  }  int lengthOfPath(int\* path, int size) {  int count = 0;  for (int i = 0; i < size; i++)  if (path[i] != -1)  count++;  return count;  }  int\* DFS\_longestPath(int\*\* graph, int size, int\* path, int last) {  int\* newPath;  int\* actualNextPath = new int[size];  for (int i = 0; i < size; i++)  actualNextPath[i] = path[i];  for (int i = 0; i < size; i++) {  if (graph[last][i] != 0 && !arrContains(path, size, i)) {  newPath = addAsLast(path, size, i);  newPath = DFS\_longestPath(graph, size, newPath, i);  if (lengthOfPath(actualNextPath, size) < lengthOfPath(newPath, size))  actualNextPath = newPath;  }  }  return actualNextPath;  }  int\*\* BFS\_deletingCycles(int\*\* graph, int size, int first)  {  Queue queue;  queue.push(first, -1);  int\* visited = new int[size];  for (int i = 0; i < size; i++)  visited[i] = -1;  visited[0] = first;  while (!queue.isEmpty()) {  for (int i = 0; i < size; i++) {  if (graph[queue.peek().val][i] != 0 && i != queue.peek().prev) {  if (arrContains(visited, size, i)) {  cout << "В текущем пути есть перекрестный элемент: " << i << endl;  excludeRing(graph, size, i);  return graph;  }  queue.push(i, queue.peek().val);  visited = addAsLast(visited, size, i);  }  }  queue.pop().val;  }  return graph;  }  int\*\* removeRings(int\*\* graph, int size)  {  int\* longestPath = new int[size];  int\* currPath = new int[size];  int\*\* startGraph = graphCopy(graph, size);  bool firstCycle = true;  while (!graphsEquals(startGraph, graph, size) || firstCycle) {  for (int i = 0; i < size; i++) {  longestPath[i] = -1;  currPath[i] = -1;  }  startGraph = graphCopy(graph, size);  for (int i = 0; i < size; i++) {  currPath[0] = i;  int\* longestPathFrom\_i = DFS\_longestPath(graph, size, currPath, i);  if (lengthOfPath(longestPath, size) < lengthOfPath(longestPathFrom\_i, size))  longestPath = longestPathFrom\_i;  }  cout << "Самый длинный путь: ";  for (int i = 0; i < size; i++) {  if (longestPath[i] != -1)  cout << longestPath[i] << " ";  }  cout << "\n";  for (int i = 0; i < size; i++) {  if (!arrContains(longestPath, size, i))  excludeRing(graph, size, i);  }  int\*\* graphWithCycles = graphCopy(graph, size);  graph = BFS\_deletingCycles(graph, size, longestPath[0]);  while (!graphsEquals(graphWithCycles, graph, size)) {  graphWithCycles = graphCopy(graph, size);  graph = BFS\_deletingCycles(graph, size, longestPath[0]);  }  firstCycle = false;  }  return graph;  } |

1. **Контрольный пример**

Приведём пример к задаче и решим его разработанным алгоритмом.

Пусть дано 11 колец, сцепленных друг с другом следующим образом (Рисунок 1).

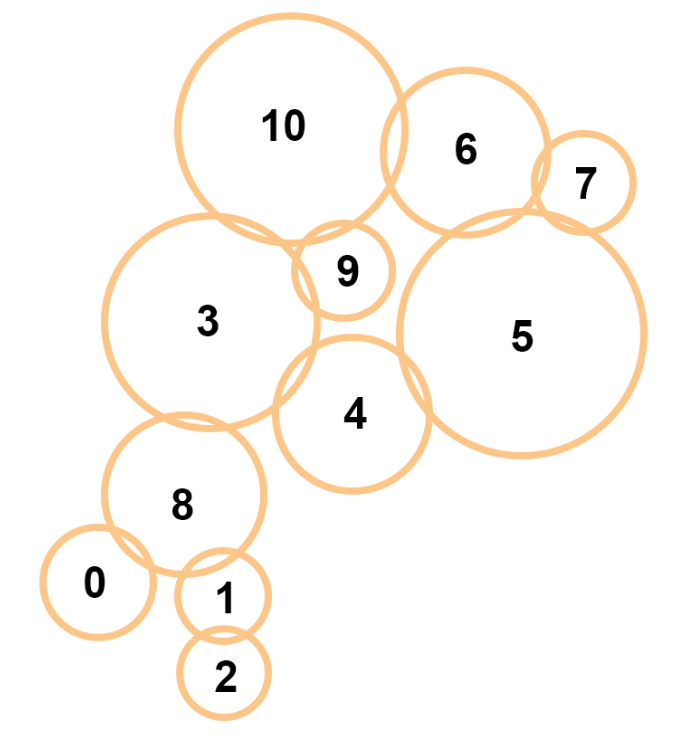


Рисунок 1 – Пример 11 сцепленных колец

Сцепленные кольца можно представить в виде графа (Рисунок 2).

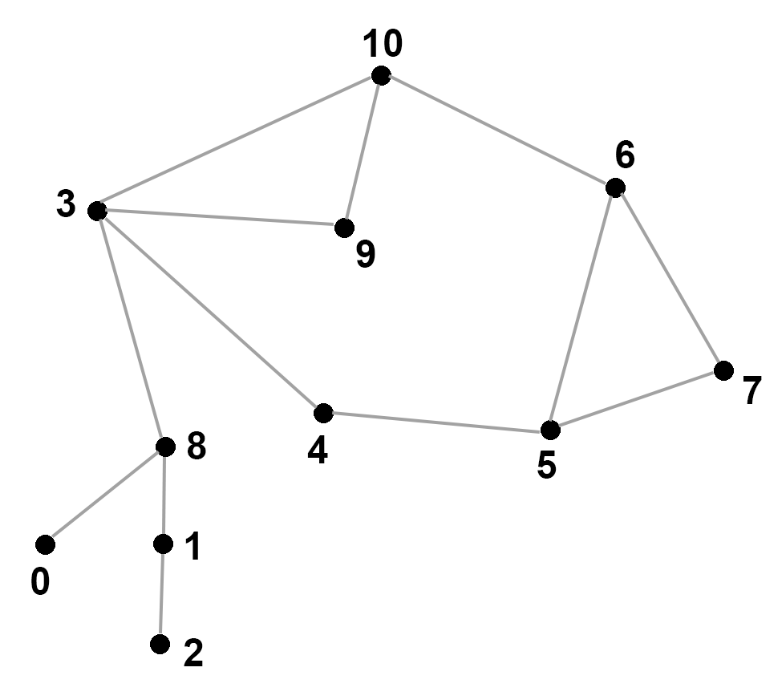


Рисунок 2 – Граф, отображающий связи 11 сцепленных колец

Граф, представленный на рисунке 2 виде матрицы смежности выглядит так (Рисунок 3).

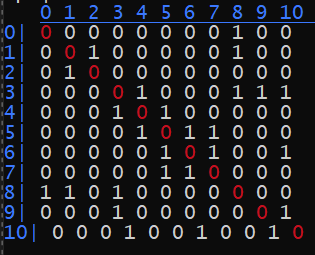


Рисунок 3 – Матрица смежности для представления графа сцепленных колец

Будем считать цепочкой незамкнутую последовательность колец, имеющую начало и конец, в которой каждое звено соединяет не более чем 2 других звена.

Текущее состояние сцепленных колец не является цепочкой, поскольку некоторые кольца связаны сразу с несколькими другими кольцами, что уже является причиной присутствия циклического соединения колец.

Применим разработанный алгоритм.

Итерация 1.

Поиск самого длинного пути звеньев. Результат представлен на рисунке 4.

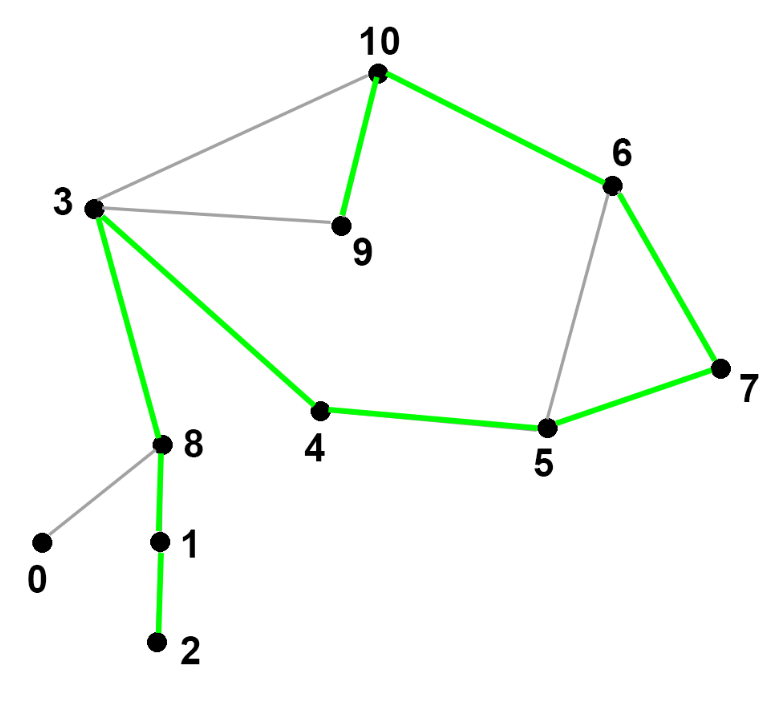


Рисунок 4 – Итерация 1, самый длинный путь

Отбросим звенья, не вошедшие в самый длиный путь. Результат представлен на рисунке 5.

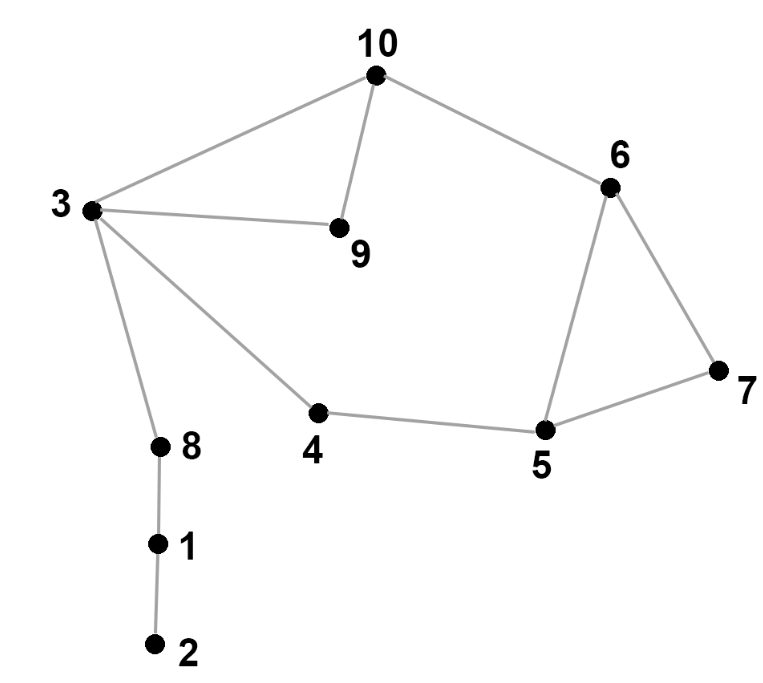


Рисунок 5 – Итерация 1, остались только вершины самого длинного пути

Применим алгоритм прохода в ширину для удаления звеньев, встречающихся во время прохода несколько раз. Начнем проход в первой вершины длиннейшего пути, т.е. со звена 2. Результат представлен на рисунке 6.

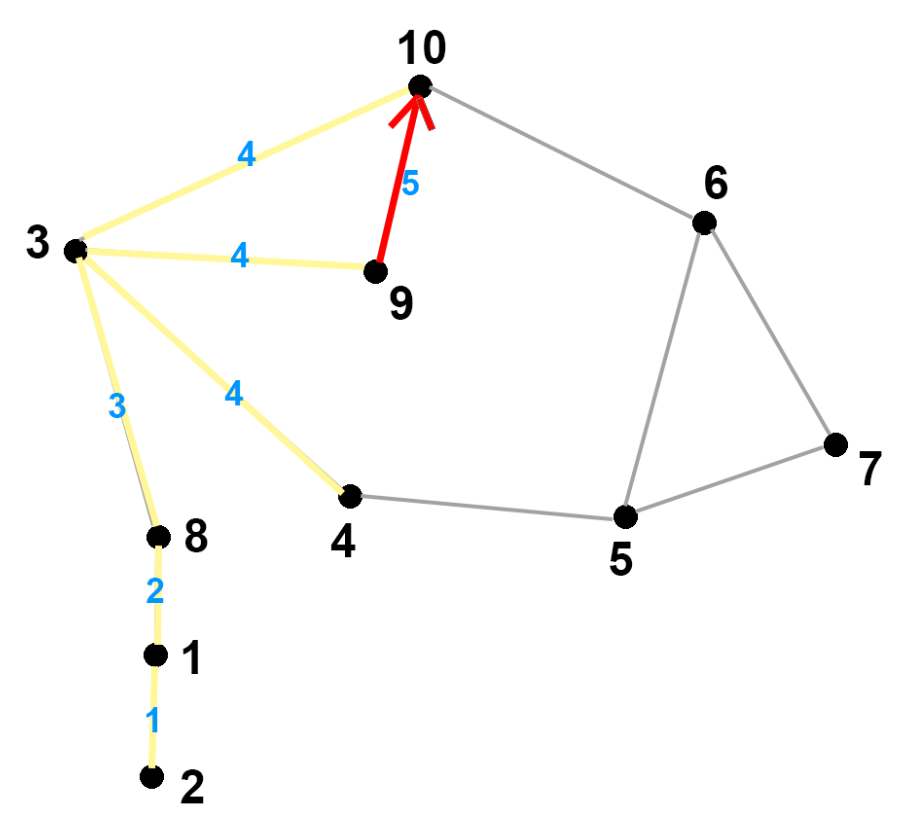


Рисунок 6 – Итерация 1, первый проход в ширину

Вершина 10 встретилась второй раз во время прохода в ширину, поэтому удалим её. Результат представлен на рисунке 7.

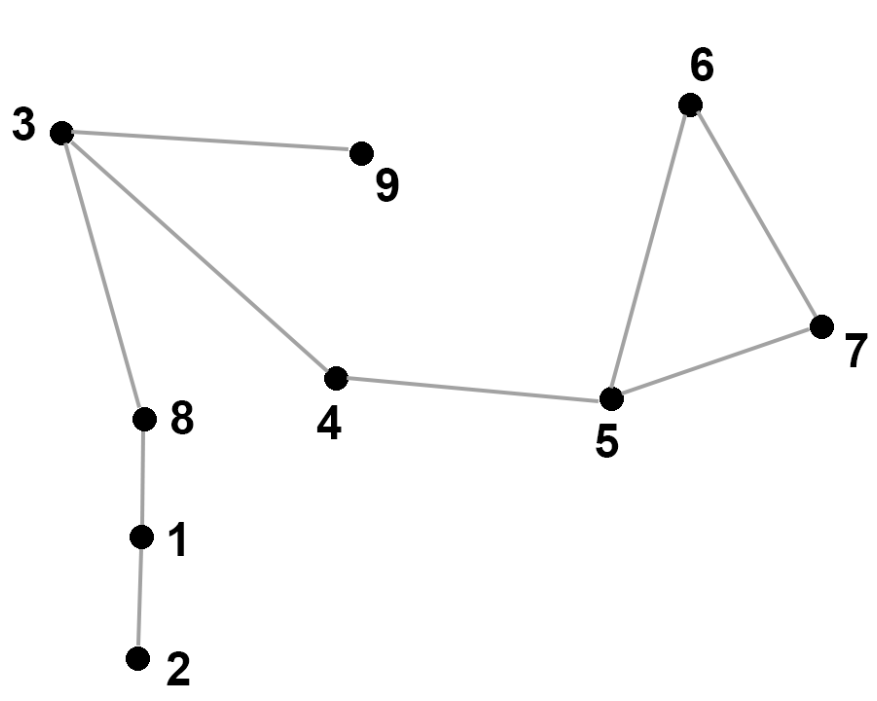


Рисунок 7 – Итерация 1, после первого прохода в ширину

Начнём проход в ширину заново. Результат представлен на рисунке 8.

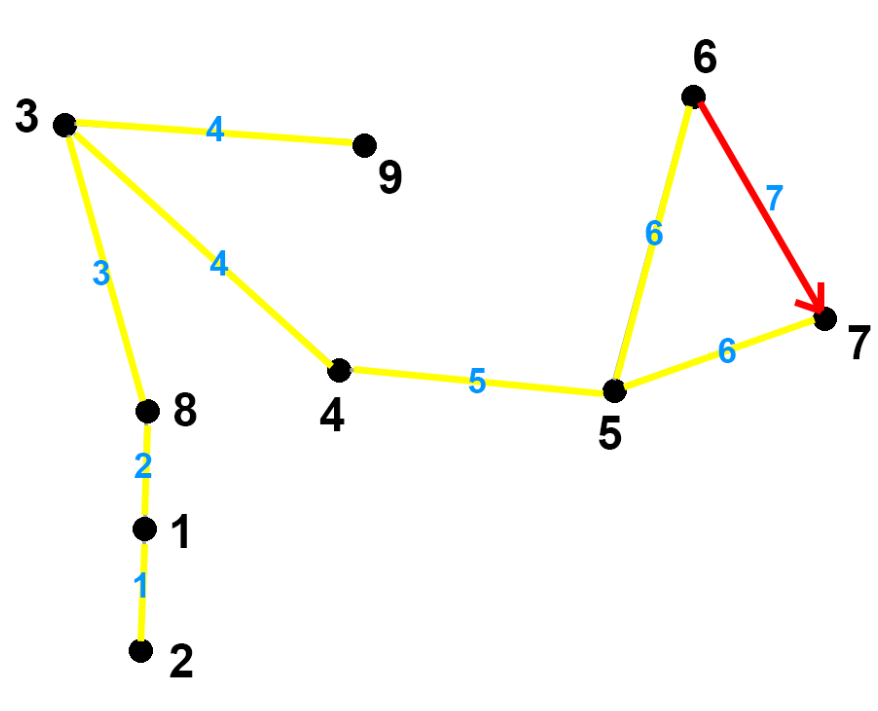


Рисунок 8 – Итерация 1, второй проход в ширину

Вершина 7 встретилась второй раз во время прохода в ширину, поэтому удалим её. Результат представлен на рисунке 9.

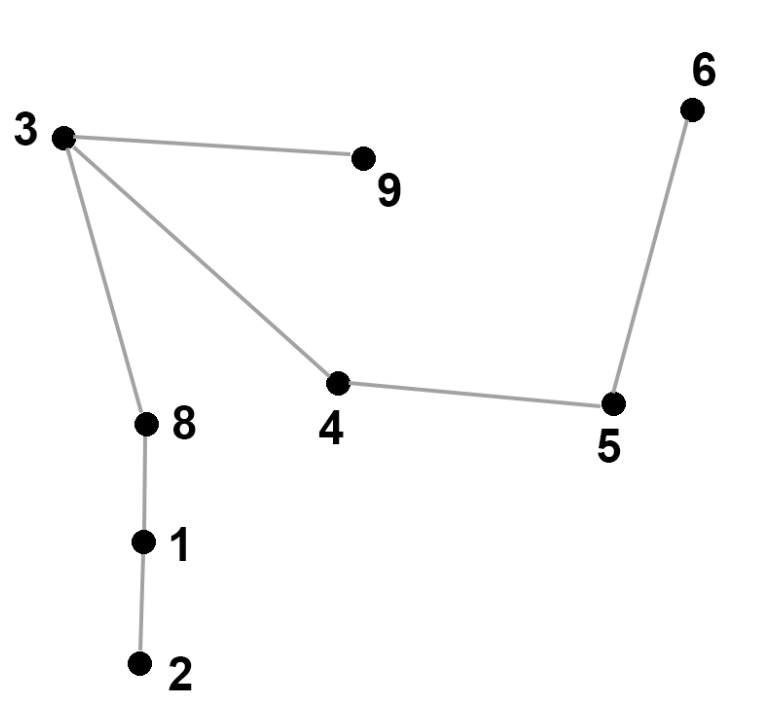


Рисунок 9 – Итерация 1, после второго прохода в ширину

Начнём проход в ширину заново. Результат представлен на рисунке 10.

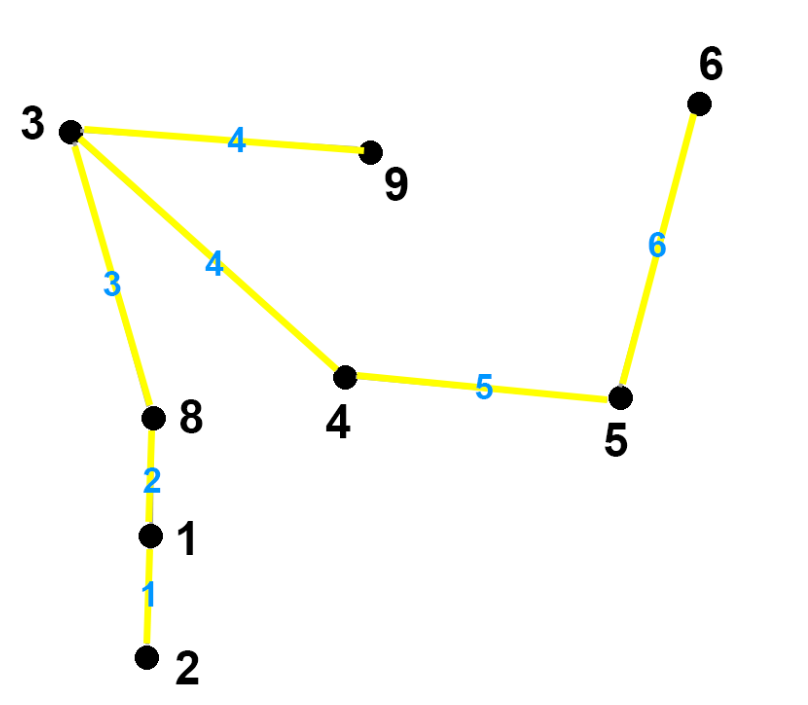


Рисунок 10 – Итерация 1, третий проход в ширину

Проход в ширину установил, что ни одна вершина не нуждается в удалении.

Итерация 1 завершена.

Граф, полученный в результате проходов в ширину на 1 итерации (рисунок 10), не является идентичным изначальному графу 1 итерации (рисунок 4). Следовательно, необходимо приступить ко 2-ой итерации.

Итерация 2.

Выполним поиск самого длинного пути звеньев. Результат представлен на рисунке 11.

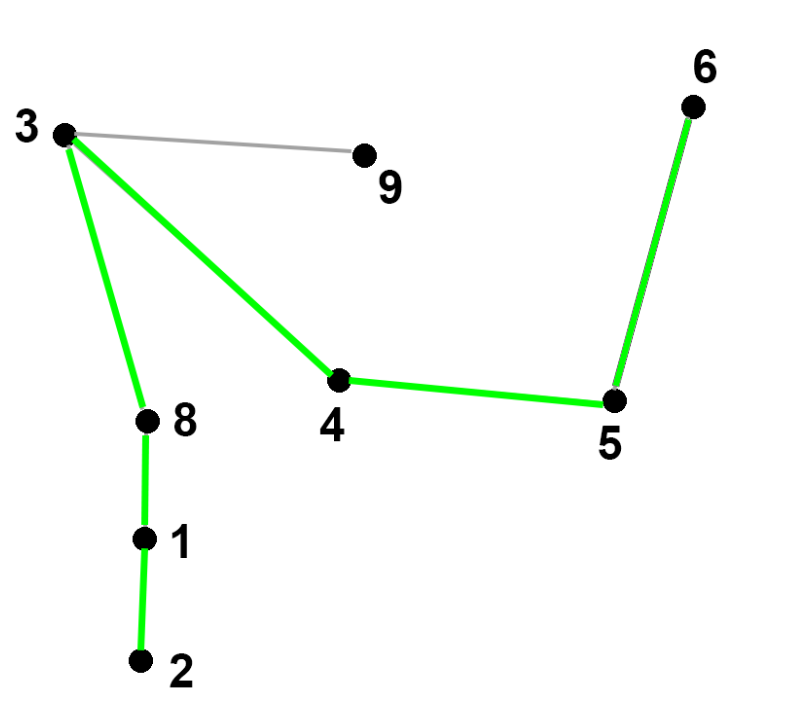


Рисунок 11 - Итерация 2, самый длинный путь

Отбросим звенья, не вошедшие в самый длиный путь. Результат представлен на рисунке 12.

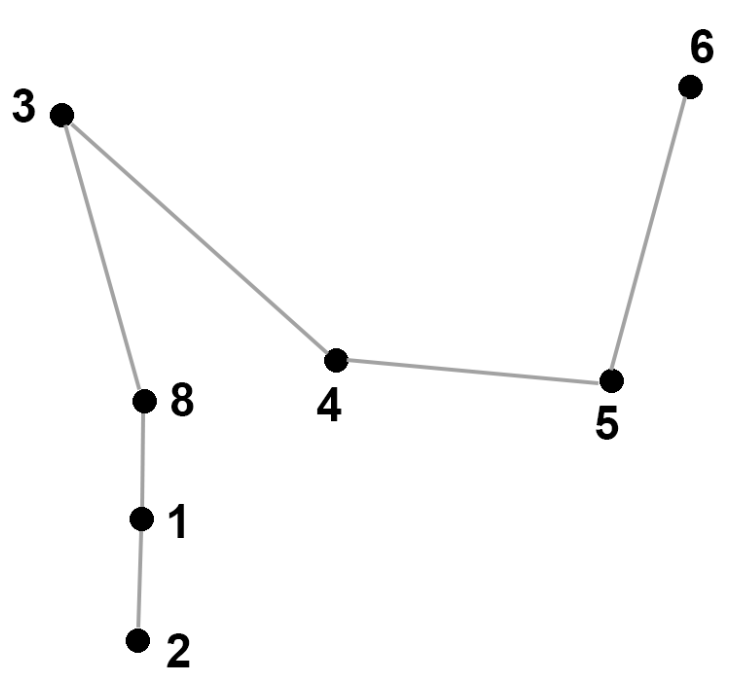


Рисунок 12 - Итерация 2, остались только вершины самого длинного пути

Применим алгоритм прохода в ширину. Начнем проход в первой вершины длиннейшего пути, т.е. со звена 2. Результат представлен на рисунке 13.

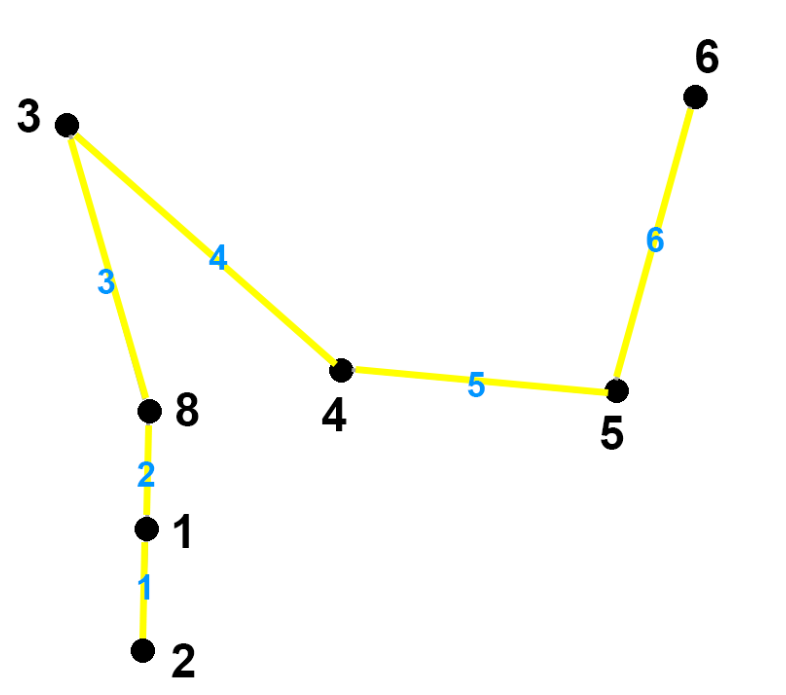


Рисунок 13 - Итерация 2, первый проход в ширину

Проход в ширину установил, что ни одна вершина не нуждается в удалении.

Итерация 2 завершена.

Граф, полученный в результате проходов в ширину на 2 итерации (рисунок 13), не является идентичным изначальному графу 2 итерации (рисунок 9). Следовательно, необходимо приступить к 3-ей итерации.

Итерация 3.

Выполним поиск самого длинного пути звеньев. Результат представлен на рисунке 14.

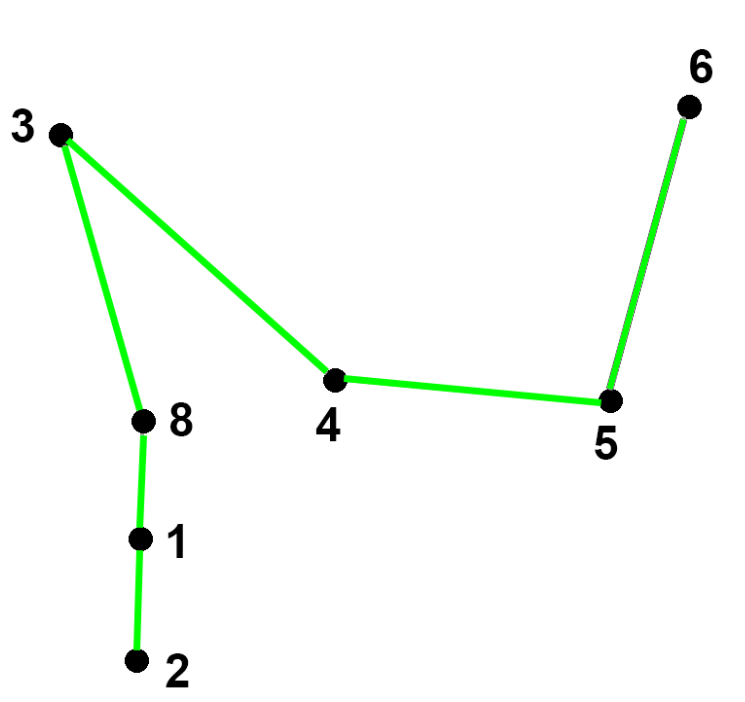


Рисунок 14 - Итерация 3, самый длинный путь

Путь охватывает все вершины, нет необходимости удалять ни одну вершину. Приступим сразу к поиску в ширину. Результат представлен на рисунке 15.

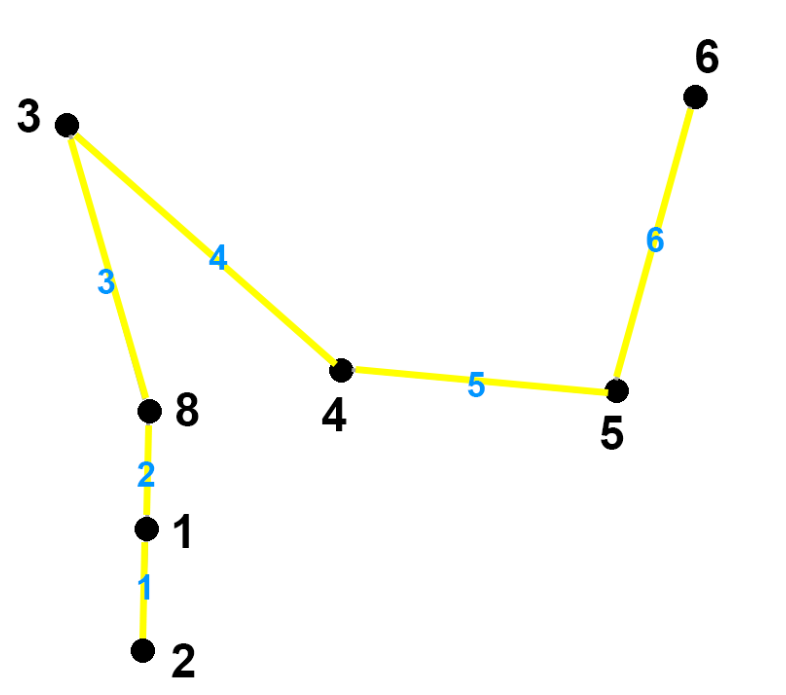


Рисунок 15 - Итерация 3, первый проход в ширину

Проход в ширину установил, что ни одна вершина не нуждается в удалении.

Итерация 3 завершена.

Граф, полученный в результате проходов в ширину на 3 итерации (рисунок 15), **является** идентичным изначальному графу 3 итерации (рисунок 13). Следовательно, задача решена и алгоритм завершает работу.

После отработки алгоритма, имеем следующее решение: граф звений изображен на рисунке 16.

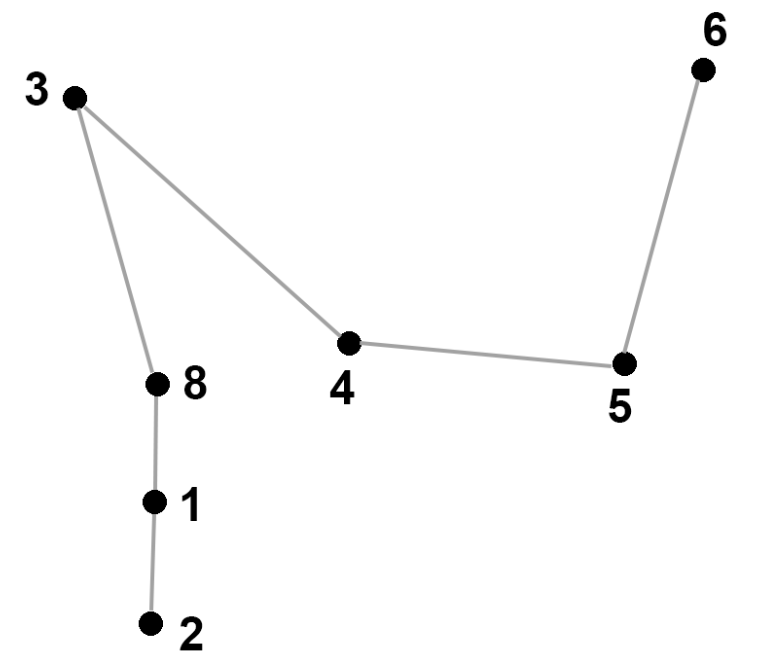


Рисунок 16 – Ответ на задачу в виде графа

В виде матрицы смежности граф представлен на рисунке 17.

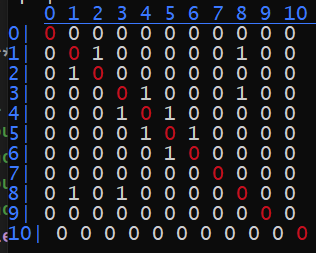
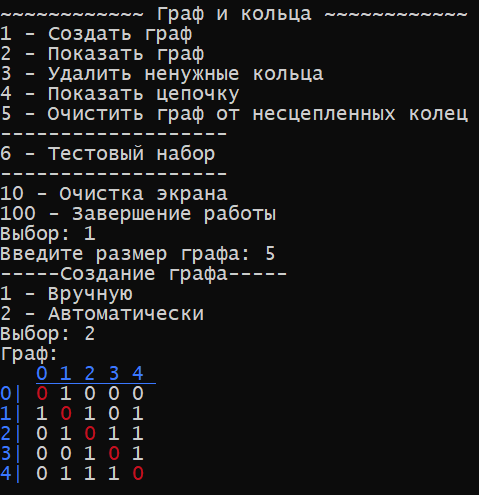
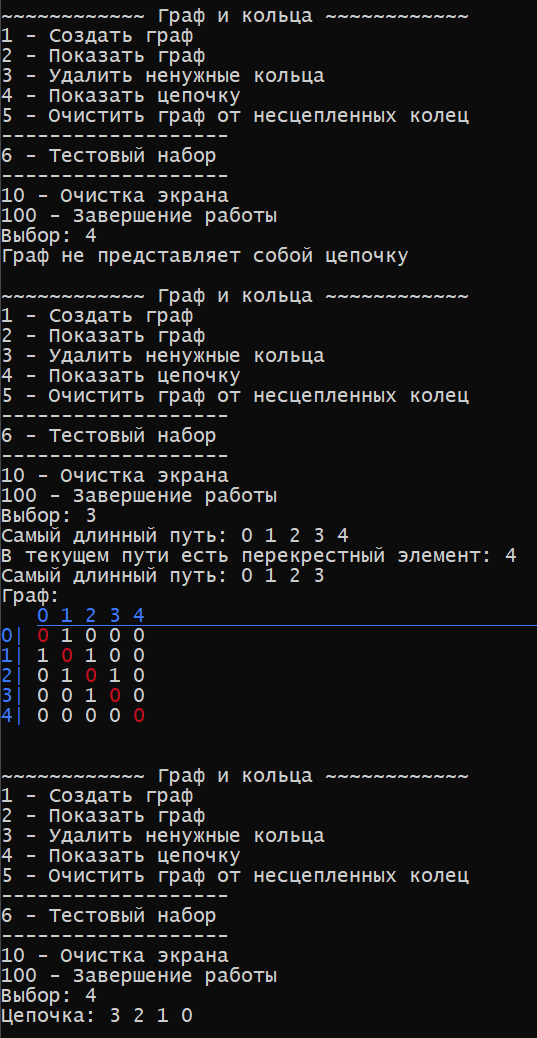


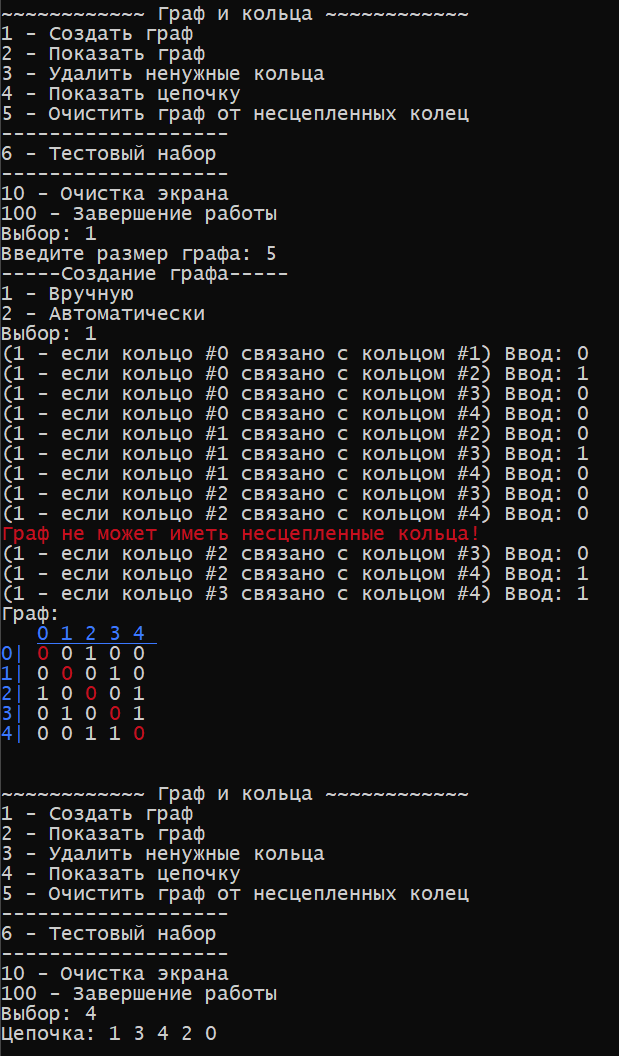
Рисунок 17 – Ответ на задачу в виде матрицы смежности

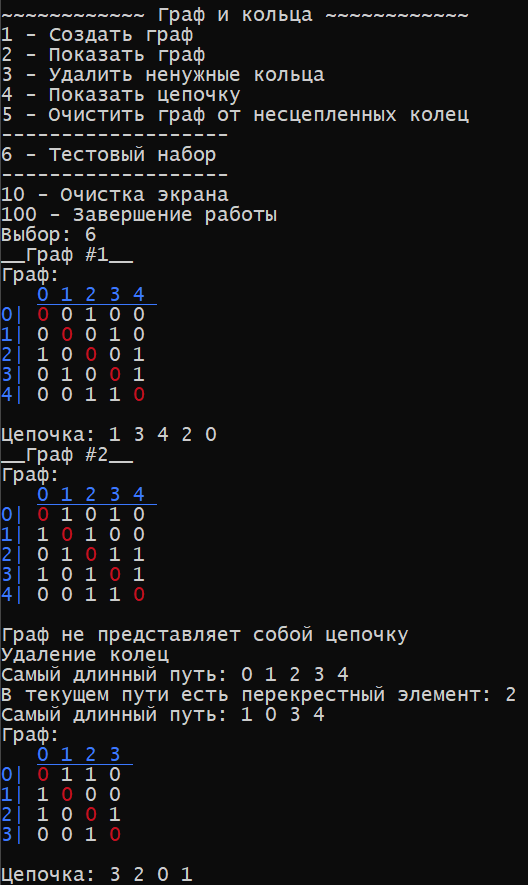
Цепочка, полученная по графу (по рисунку 16 или по матрице смежности, рисунок 17): 6, 5, 4 ,3, 8, 1, 2.

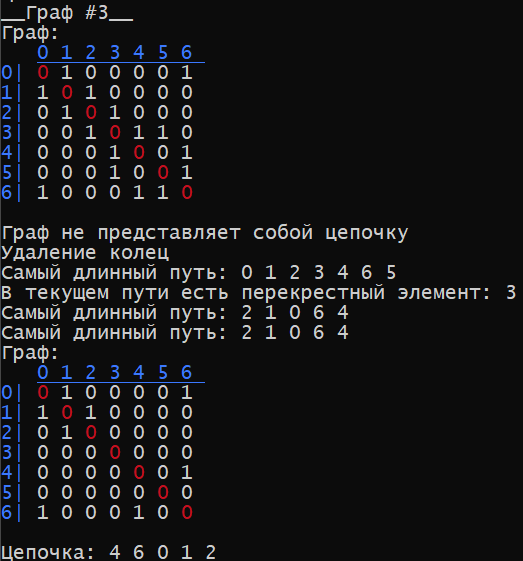
1. **Результаты работы программы**

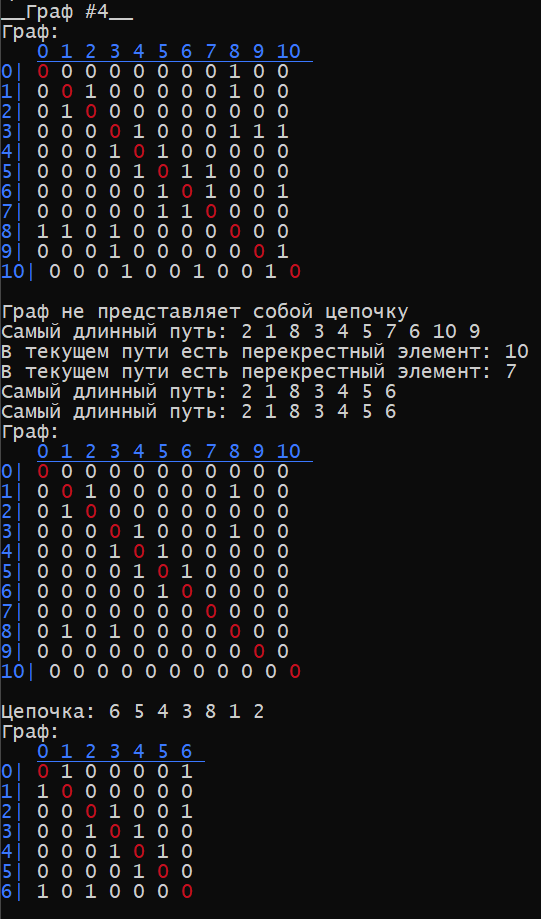












1. **Вывод**

Изучена теория графов и получены практические навыки их использования. Применены алгоритмы прохода графа в ширину и в глубину для решения поставленной задачи.