Федеральное агентство по образованию

ФГБОУ ВО Уфимский государственный авиационный технический

университет

Кафедра ВМиК

Расчетно-графическая работа

По дисциплине «Алгоритмы и структуры данных»

Вариант 17

Выполнил: студент группы ПРО-222 Хусамов А.Р.

Проверил: Верхотурова Г.Н.

Уфа 2020

**Задание:** Орграф, заданный с помощью списков смежности, проверить на наличие

циклов и при их наличии вывести каждый цикл в виде вершин циклического пути.

**Определение:**

***Список******смежности*** — один из способов представления графа в виде коллекции списков вершин. Каждой вершине графа соответствует список, состоящий из «соседей» этой вершины

***Цикл*** – замкнутая цепь в графе.

***Цепь*** – маршрут, в котором все ребра различны.

***Маршрут*** – чередующая последовательность вершин и ребер графа.

**Алгоритм нахождения циклов в графе:**

При обходе используем три цвета вершин:

* “white” – не посещенная,
* “grey” – в процессе обработки,
* “black” – обработанная вершина.

Цикл существует, если поиск в глубину обнаруживает ребро, конец которого покрашен в цвет “grey”.

Условные обозначения:

u – текущая вершина

v – смежная вершина с текущей u вершиной

parent [u] - хранит родителя вершины u

disc [] - для хранения времени обнаружения вершин

low [u] – указывает на самую раннюю посещенную вершину(вершину с минимальным временем обнаружения), достижимую из поддерева с корнем u

1. Запустим обход в глубину из произвольной вершины графа.
2. Пометим текущую вершину как посещённую и отметим у неё disc и low. (время обнаружения вершины и самую раннюю посещённую вершину)
3. Пройдём все вершины смежные с текущей и если она не была посещена, то

parent[v] = u (укажем вершину u, как родителя для вершины v). Вызовем рекурсию, но уже вершиной будет v и снова перейдём ко 2 пункту. (т. е. будем идти в глубь графа)

1. Присваиваем для low[u] минимальную вершину, среди ранее посещённых у вершин u и v. И если самая ранняя посещённая вершина у вершины v будет больше, чем время обнаружения у вершины u, то ребро u – v будет являться мостом в графе. Выведем мост в ListBox.
2. Если же смежная вершина(v) была уже посещена, и она не равна родителю у текущей вершины вершины(parent[u]), то для low[u] присваиваем наименьшее значение среди low[u] и временем обнаружения у смежной вершины(disc[v]).

**Входные данные:**

* Построенный пользователем невзвешенный неориентированный граф
* dataGridView\_graph[i, j] - исходная матрица

**Вспомогательные данные:**

* copy\_adjacencymatrix – копия исходной матрицы
* adj – лист, хранящийся пути из вершин
* bool[i] visited – посещена ли вершина
* int[] disc – сохраняет время посещённых вершин
* int[] parent – сохраняет родительские вершины в дереве DFS
* int[] low – самую раннюю посещенную вершину

**Выходные данные:**

* ListBox\_result – поле с выводом мостов графа
* В случае отсутсвия мостов графа будет выведено соответствующее сообщение

**Код поиска в ширину:**

public void bfs(int position)

{

countBFS = 0;

List<int> list = new List<int>(); // Вершины

Queue<int> queue = new Queue<int>(); //

list.Add(position); // Добавляем в список стартовую позицию

for (int i = 0; i < n; ++i) // Добавляем в очередь все смежные пути

if (dataGridView\_graph[position, i].Value != null)

queue.Enqueue(i + 1);

while (queue.Count != 0)

{

bool repeated = false;

int v = queue.Dequeue(); // Получаем первое число из очереди и убираем его

foreach (int i in list) // Если полученный элемент уже есть в списке, то идем к следующему

if (i == v)

{

repeated = true;

break;

}

if (repeated)

continue;

list.Add(v); // Иначе добавляем это число в список

for (int i = 0; i < n; ++i)

if (dataGridView\_graph[v, i].Value != null)

queue.Enqueue(i + 1);

}

foreach (int i in list)

{

++countBFS;

}

}

**Код функции add\_list:**

public void add\_list(int v)

{

adj = new List<int>[v];

for (int i = 0; i < v; ++i)

adj[i] = new List<int>();

}

**Код функции отрисовки ребра:**

public void setcolorline(Color color, int v1, int v2)

{

Pen pen = new Pen(color, 3);

Point pt1 = new Point(circles[v1].x + circles[v1].rad, circles[v1].y + circles[v1].rad);

Point pt2 = new Point(circles[v2].x + circles[v2].rad, circles[v2].y + circles[v2].rad);

panel\_drawing\_graph.CreateGraphics().DrawLine(pen, pt1, pt2);

}

**Код нажатия на кнопку поиска мостов в графе:**

private void btn\_findbridge\_Click(object sender, EventArgs e)

{

listBox\_result.Items.Clear();

bfs(1);

if (countBFS != n)

{

MessageBox.Show("В графе нет мостов, так как граф несвязный");

return;

}

int[,] copy\_adjacencymatrix = new int[n, n];

for (int i = 1; i <= n; ++i) // Копируем матрицу смежности из dataGridView\_graph

for (int j = 1; j < n; ++j)

copy\_adjacencymatrix[i - 1, j] =

Convert.ToInt32(dataGridView\_graph[i, j].Value);

add\_list(n);

for (int i = 0; i < n; ++i) // Находим все пути из вершины

{

for (int j = 0; j < n; ++j)

{

if (j == i && i != 0)

break;

if (copy\_adjacencymatrix[i, j] == 1)

{

adj[i].Add(j);

adj[j].Add(i);

setcolorline(Color.Black, i, j); // Рисуем линию

}

}

}

// Пометить все вершины как непосещенные

bool[] visited = new bool[n];

int[] disc = new int[n];

int[] low = new int[n];

int[] parent = new int[n];

for (int i = 0; i < n; i++)

{

parent[i] = -1;

visited[i] = false;

}

DFSwithbridges(0, visited, disc, low, parent); // Вызываем функция поиска моста в графе

if (listBox\_result.Items.Count == 0) // Если мостов нет в графе

MessageBox.Show("В заданном графе мостов нет");

}

**Код функции поиска мостов в графе:**

void DFSwithbridges(int u, bool[] visited, int[] disc, int[] low, int[] parent)

{

visited[u] = true; // Пометить текущую вершину как посещенную

disc[u] = low[u] = ++time; // Инициализируем время обнаружения и низкое значение

foreach (int i in adj[u]) // Проходим все вершины, примыкающие к текущей

{

int v = i; // v является текущим смежным с u

if (!visited[v])

{

parent[v] = u;

DFSwithbridges(v, visited, disc, low, parent);

// Проверяем, есть ли у поддерева с корнем v связь с одним из предков u

low[u] = Math.Min(low[u], low[v]);

// Если самая низкая вершина достижима из поддерева

// v находится ниже u в дереве DFS, тогда u-v будет мост

if (low[v] > disc[u])

{ // Выводим мост в ListBox

listBox\_result.Items.Add((u + 1).ToString() + " -> " +

(v + 1).ToString());

setcolorline(Color.Red, u, v); // Рисуем линию

}

}

else if (v != parent[u])

low[u] = Math.Min(low[u], disc[v]); // Обновление нижнего значения для u

}

}

**Анализ порядка сложности алгоритма по управляющим структурам:**

**Функция DFSwithbridges:**

void DFSwithbridges(...)

{

...

foreach (int i in adj[u])

{

...

if (!visited[v])

O(n+m)

{

O(n+m)

...

DFSwithbridges(...);

...

}

}

}

Подсчитаем сложность данного алгоритма. Поскольку данный алгоритм основан на рекурсивном поиске в глубину и мы обходим каждого «соседа» каждого узла, игнорируя тех, которых посещали ранее, мы имеем время выполнения, равное O(n+ m), где n – общее кол-во вершин, m – общее количество граней(ребер).

**Функции BFS:**

public void bfs(int position)

{

...

for (int i = 0; i < n; ++i)// Добавляем в очередь все смежные пути

O(N)

...

while (queue.Count != 0)

{

...

O(N2)

foreach (int i in list)

O(N)

O(N2)

...

for (int i = 0; i < n; ++i)

O(N)

...

}

foreach (int i in list)

O(N)

...

}

Сложность функции = O(N2)

**Функции отрисовки линии:**

public void setcolorline(Color color, int v1, int v2)

{

O(I)

...

}

Сложность функции = O(I)

**Функции add\_list**

public void add\_list(int n)

{

...

for (int i = 0; i < n; ++i)

O(N)

O(N)

...

}

Сложность функции = O(N)

**События нажатия на кнопку “Найти мост” btn\_findbridge\_Click:**

private void btn\_findbridge\_Click(object sender, EventArgs e)

{

O(N2)

...

bfs(1);

...

for (int i = 1; i <= n; ++i)

// Копируем матрицу смежности из dataGridView\_graph

O(N2)

for (int j = 1; j < n; ++j)

...

O(N)

O(N)

add\_list(n);

for (int i = 0; i < n; ++i) // Находим все пути из вершины

{

for (int j = 0; j < n; ++j)

{

...

if (…)

O(N2)

{

O(N)

...

O(I)

setcolorline(…); // Рисуем линию

}

}

}

...

for (int i = 0; i < n; i++)

{

O(N)

...

}

O(n+m)

DFSwithbridges(…); // Вызываем функция поиска моста в графе

...

}

Сложность события нажатия на кнопку:

* при n < 5 в худшем случае = O(n²), в лучшем = O(n + m)
* при n ≥ 5 в худшем случае = O(n + m), в лучшем = O(n²)

n – общее кол-во вершин, m – общее количество граней(ребер).

**Пример работы программы:**

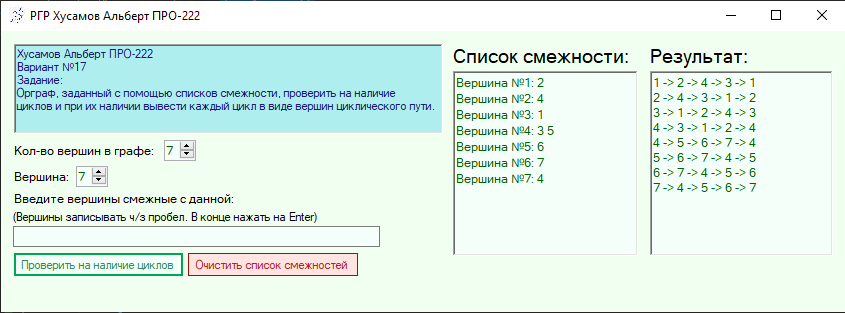


Рис. Демонстрация работы программы

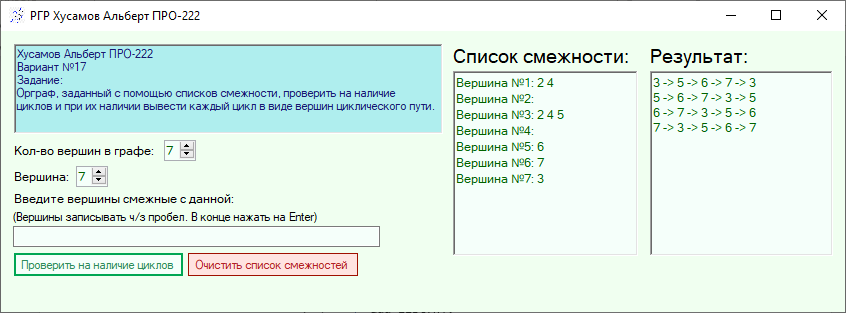


Рис. Демонстрация работы программы

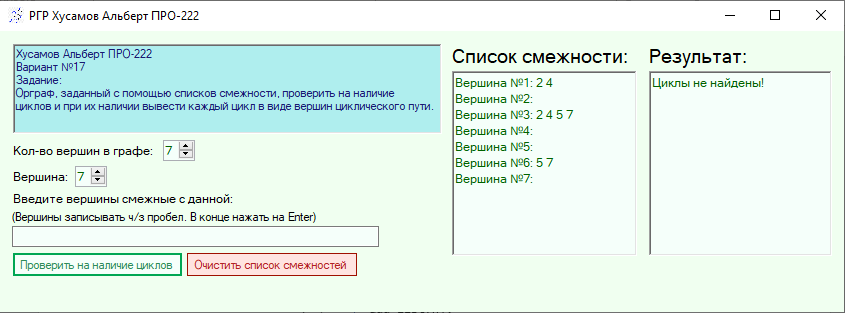


Рис. Демонстрация работы программы

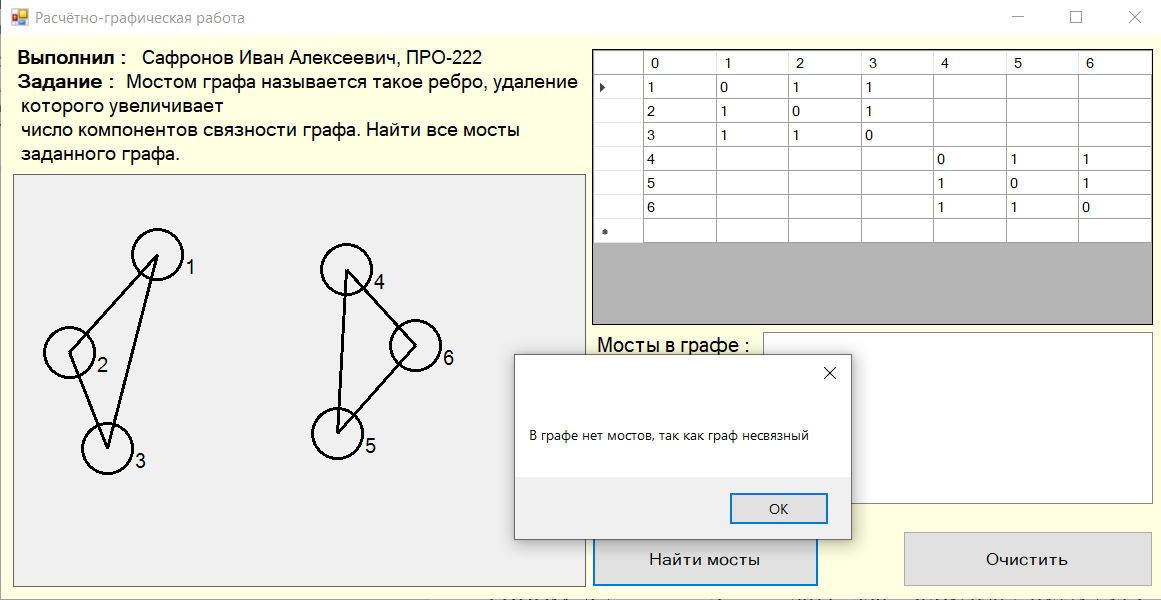


Рис. Демонстрация работы программы

СПИСОК ЛИТЕРАТУРЫ

1. Алексеев, В.Е. Графы и алгоритмы. Структуры данных. Модели вычислений. Москва: БИНОМ, Лаборатория Знаний, 2011,–320 с. ISBN 978-5-94774-543-6.

2.Ф.А. Новиков. Дискретная математика для программистов. СПб: Питер, 2000.

3. Б.С. Хусаинов. Структуры и алгоритмы обработки данных. Примеры на языке Си: Учебное пособие.– М.: Финансы и статистика, 2004.– 464 с.–ISBN 5-279-02775-8.

4. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн Алгоритмы: построение и анализ = Introduction to Algorithms. — 2-е изд. — М.: «Вильямс», 2006. — С. 1296.

5. Майника Э. Алгоритмы оптимизации на сетях и графах. 1981.

6. Евстигнеев В.А. Применение теории графов в программировании. 1985.

7. Райли Д. Абстракция и структуры данных. Вводный курс. 1993.

8. Вирт, Н. Алгоритмы и структуры данных. Новая версия для Оберона [электронный ресурс]. Москва: ДМК ПРЕСС, 2010.–272 с.– ISBN 978-5-94074-584-6.

9. Липский В. Комбинаторика для программистов. 1988.

10. Свами М., Тхуласираман К. Графы, сети и алгоритмы. 1984.

11.Ф.Препарата, М.Шеймос. Вычислительная геометрия. М.: «Мир», 1989.

12. Ахо А.В., Хопкрофт Д., Ульман Д. «Структуры данных и алгоритмы». М.: Вильямс, 2000

13. В.М. Бондарев, В.И. Рублинецкий, Е.Г. Качно. "Основы программирования". Харьков, "Фолио", Ростов-на-Дону, "Феникс". 1997.

14. Уильям Топп, Уильям Форд. Структуры данных в C++. 1999.

15. С.Д. Кондратьева. "Введение в структуры". Москва, МГТУ им. Баумана, 2000.