Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по курсу «Программирование»

на тему «Реализация алгоритма выделения компонент связности орграфа, используя поиск в глубину»

Выполнил студент группы 20ВВ3:

Мирясов Н.А.

Приняли:

Митрохин М.А.

Юрова О.В.

Пенза 2021

Содержание

[Реферат 3](#_Toc90922390)

[Введение 4](#_Toc90922391)

[1 Постановка задачи 5](#_Toc90922392)

[2 Теоретическая часть 6](#_Toc90922393)

[3 Описание алгоритма программы 8](#_Toc90922394)

[4 Описание программы 16](#_Toc90922395)

[5 Тестирование 24](#_Toc90922396)

[6 Ручной расчёт задачи 31](#_Toc90922397)

[Заключение 34](#_Toc90922398)

[Список литературы 35](#_Toc90922399)

[Приложение A Листинг программы. 36](#_Toc90922400)

# Реферат

Отчёт 48 стр., 12 рис.

Граф, теория графов, орграф, достижимость, поиск в глубину, компонента.

Цель исследования – разработка программы, способная выделять

компоненты сильной связности орграфа, использую алгоритм поиска в

глубину.

В работе рассмотрены правила поиска в глубину, на основе которых

находится компонента сильной связности орграфа. Установлено, что с

помощью данного алгоритма можно выделить компоненту сильной связности

подграфа орграф, которого несвязен, слабо связан или сильно связан.

# Введение

Алгоритм поиска (или обхода) в глубину (англ. depth-firstsearch, DFS)

позволяет построить обход ориентированного или неориентированного графа,

при котором посещаются все вершины, доступные из начальной вершины.[1]

Отличие поиска в глубину от поиска в ширину заключается в том, что (в

случае неориентированного графа) результатом алгоритма поиска в глубину

является некоторый маршрут, следуя которому можно обойти последовательно все вершины графа, доступные из начальной вершины. Этим он принципиально отличается от поиска в ширину, где одновременно обрабатывается множество вершин, в поиске в глубину в каждый момент исполнения алгоритма обрабатывается только одна вершина.[2]

С другой стороны, поиск в глубину не находит кратчайших путей, зато он применим в ситуациях, когда граф неизвестен целиком, а исследуется каким-то автоматизированным устройством.

Если же граф ориентированный, то поиск в глубину строит дерево путей из начальной вершины во все доступные из нее.

В качестве среды разработки мною была выбрана среда IntelliJ IDEA Community Edition 2021.2.1, язык программирования – Java.

Целью данной курсовой работы является разработка программы на языке Java, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм поиска в глубину, осуществляющий поиск компоненты сильной связности орграфа.

# 

# 1 Постановка задачи

Требуется разработать программу, которая выделит компоненты сильной связности орграфа, используя алгоритм поиска в глубину. Исходный граф в программе должен задаваться матрицей смежности, причем при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После обработки этих данных на экран должна выводиться матрица смежности орграфа, вид орграфа и все компоненты связности орграфа. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь.

# 2 Теоретическая часть

Граф G (см. рис. 1) задается множеством вершин X1, X2, ..., Xn. и

множеством ребер, соединяющих между собой определенные вершины. Ребра из множества А ориентированы, что показывается стрелкой, которая указывает достижимость данной вершины, граф с такими ребрами называется ориентированным графом.[3]

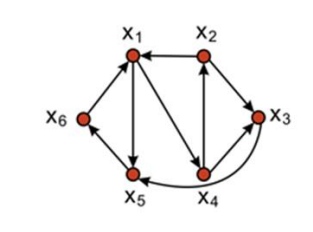


Рисунок 1 – Пример орграфа

При представлении графа матрицей смежности информация о ребрах

графа хранится в квадратной матрице, где присутствие пути из одной вершины в другую обозначается единицей, иначе нулем.

Существует много алгоритмов на графах, в основе которых лежит систематический перебор вершин графа, такой, что каждая вершина графа просматривается только один раз, и переход от одной вершины к другой осуществляется по ребрам графа. Остановимся на одном из двух стандартных методах такого перебора - поиск в глубину.

Пусть задан орграф G= (X, a) и вершина X1, с которой начинается обход. После посещения вершины X1, следующей за ней будет посещена смежная с X1 вершиной. Далее, эта процедура повториться для вершин смежных с вершинами из множества q, за исключением вершины X1, т.к. она уже была посещена. Так, продолжая обходить в глубину, алгоритм обойдет все доступные из X1 вершины орграфа G. Алгоритм прекращает свою работу после обхода всех вершин графа, либо в случае выполнения наличествующего условия.

Орграф называется сильно связным, если любые две его вершины сильно связаны. Две вершины X1 и X2 любого орграфа сильно связаны, если существует ориентированный путь из X1 в X2 и ориентированный путь из X2 в X1. Компонентами сильной связности орграфа называются его максимальные по включению сильно связные подграфы.

Любая вершина орграфа сильно связана сама с собой.

# 3 Описание алгоритма программы

Для программной реализации алгоритма понадобиться четыре массива: vis (int) – для хранения информации о посещенных вершинах (1 или 0), component (int), в который будут заноситься найденные компоненты, и дублироваться в массив component\_ob (int) для дальнейшего сравнивания, vDelete (int), в который будут заносится вершины не входящие в компоненту и после удаляться из неѐ и vSkip (int), в котором будут храниться вершины, которые не нужно проверять. Итак, имеется граф G = (V, E). Каждая из вершин входящих во множество V, изначально отмечена как не посещенная, т. е. элементам массива vis присвоено значение 0.

В качестве исходного пункта выбирается вершина s и ей приписывается, что она посещена: vis[s]=1. И в component[n] заносится номер вершины. Далее, находятся соседняя вершина (в которую есть ребро из s) и поочередно исследуя вглубь, выявляются все вершины. Так как из вершин существует несколько путей, то возможно такое, когда достигнута последняя вершина обхода вглубь, но не все вершины были пройдены и помечены в vis. Поэтому алгоритм рекурсивен и будет произведен откат назад к предыдущей вершине, для того чтобы пройти все вершины и component[n] занести номера этих вершин.

Далее происходит обработка на связность графа. Если запуск поиска хотя бы из одной вершины показал достижимость всех остальных, то орграф является связным и происходит выявление степени его связности.

Обнуляются все векторы и происходит новый обход из вершины s=0. В результате обхода в component\_ob заносятся значения component и происходит опять обнуление всех векторов, кроме component\_ob в котором храниться компонента вершины 0. Далее происходит снова обход уже и последующих вершин, и значения component сравниваются со значениями component\_ob. Если они равны, то к переменной NumberOfAdjacentVertices прибавляется 1. После NumberOfAdjacentVertices сравнивается с переменной str, в которой хранится количество вершин и если они равно, то переменная connectivityCounter увеличивается на 1. Пройдя все вершины и сравнив их с component, переменная connectivityCounter сравнивается с (str-1). Если они равны, то орграф сильно связный и на экран выводится компонента сильной связности вершины 0. Иначе граф слабо связный.

Если граф слабо связный или несвязный происходит обнуление всех векторов. Далее происходит снова обход. В качестве исходной вершины берется 0. В результате обхода в component\_ob заносятся значения component. А в переменную connectivityCounter количество этих значений из переменной n. После этого происходит снова обход и значения component сравниваются со значениями component\_ob. Если они равны, то к переменной NumberOfAdjacentVertices прибавляется 1. Далее происходит сравнение, если connectivityCounter = n то в вектор vSkip заносится 1 в индекс вершины, из которой был последний запуск. Иначе в вектор vDelete заносится эта вершина, и переменная s увеличивается на 1. После всех обходов последующих вершин, если s больше 1, то происходит удаление вершин из component\_ob, которые хранятся в вектор vDelete. Далее выводится component\_ob.

Если в векторе vSkip есть не помеченные вершины, то обход начинается снова, но уже из этих вершин. Алгоритм продолжается до тех пор, пока все компоненты орграфа не будут найдены.

Ниже представлен псевдокод функции Component(), DFS() и частично main().

**main()**

1.для i=0 пока i<str делать i=i+1

2. вызвать функцию DFS(i,matrix,str,vis,component) для i

3. если значение функции (Connectivity(vis, str) == 1)

4.s = s+1

5.конец условия

6. вызвать функцию toNull(vis, str);

7.вызвать функцию toNull(component,str);

8.n = 0;

9.конец цикла

10. вызвать функцию toNull(vis, str);

11.вызвать функцию toNull(component,str);

12.n = 0;

13.если s>0

14. вывод " Орграф связный"

15. вызвать функцию DFS(0,matrix,str,vis,component) для 0

16. для i = 0 пока i < n+1 делать i = i+1

17.component\_ob[i] = component[i];

18.конец цикла

19.connectivityCounter = n

20. вызвать функцию toNull(vis,str);

21.вызвать функцию toNull(component,str);

22.n = 0;

23. для m = 1 пока m < str делать m = m+1

24.вызвать функцию DFS(m,matrix,str,vis,component) для m

25. для i=0 пока i<connectivityCounter+1 делать i=i+1

26. для j=0 пока j<n+1 делать j=j+1

27.если component\_ob[i] == component[j]

28.f = f+1

29.конец условия и конец двух циклов

30.если f == str

31. g = g+1

32. f = 0

33.n = 0

34. вызвать функцию toNull(vis,str);

35. вызвать функцию toNull(component,str);

36. конец условия и цикла

37.если g == (str-1)

38. вывод " Орграф сильно связан "

39. вывод " Компонента сильной связности орграфа:"

40. для i=0 пока i<connectivityCounter+1 делать i=i+1

41. вывод component\_ob[i]

42.конец условия и цикла

43. иначе

44.connectivityCounter = 0

45.n = 0

46. вызвать функцию toNull(vis,str)

47. вызвать функцию toNull(component,str)

48. вызвать функцию toNull(component\_ob,str)

49. вывод " Орграф слабо связный "

50. вывод "Компоненты сильной связности орграфа:"

51.вызвать функцию

Component(matrix,str,component,v\_delete,v\_propysk,component\_ob,vis)

52.конец условия

53.конец условия

54. иначе

55.connectivityCounter = 0;

56.n = 0;

57. вызвать функцию toNull(component\_ob,str)

58. вывод " Орграф несвязный "

59. вывод "Компоненты сильной связности орграфа:"

60. вызвать функцию

Component(matrix,str,component,v\_delete,v\_propysk,component\_ob,vis)

61.конец условия

**Component()**

1.f = 0;

2. для i=0 пока i<str делать i=i+1

3. если v\_propysk[i] == 0

4.вызвать функцию DFS(i,matrix,str,vis,component) для i

5.r = n

6.connectivityCounter = n

7.s = 0

8. для j=0 пока j<connectivityCounter+1 делать j=j+1

9.component\_ob[j] = component[j];

10.конец цикла

11. вызвать функцию toNull(vis,str)

12. вызвать функцию toNull(component,str)

13.n = 0

14. для j=0 пока j<str делать j=j+1

15. вызвать функцию DFS(j,matrix,str,vis,component) для j

16. для m=0 пока m<connectivityCounter+1 делать m=m+1

17. для t=0 пока t<n+1 делать t=t+1

18.если component\_ob[m] == component[t]

19.f = f+1

20. конец условия и двух циклов

21.если n == connectivityCounter

22.если f == (connectivityCounter+1)

23.v\_propysk[j] = 1

24.конец двух условий

25.иначе

26.v\_delete[s] = j

27.s = s+1

28.конец условия

29.n = 0

30. вызвать функцию toNull(vis,str)

31. вызвать функцию toNull(component,str)

32.f = 0

33.конец цикла

34.если s != 0

35. для q=0 пока q<s делать q=q+1

36. для z=0 пока z<r+1 делать z=z+1

37.если component\_ob[z] == v\_delete[q]

38.component\_ob[z] = 0;

39. для x=z пока x<r+1 делать z=z+1

40.если x == r

41.component\_ob[x] = 0

42.конец условия

43.иначе

44.component\_ob[x] =

component\_ob[x+1];

45.конец условия и цикла

46.r =r – 1

47.конец условия и двух циклов

48.конец условия

49. вывод " Компонента "

50. вывод " : "

51. для y=0 пока y<r+1 делать y=y+1

52. вывод component\_ob[y]

53.конец цикла

54.NumberOfComponent = NumberOfComponent + 1

55.connectivityCounter = 0;

56.toNull(vis,str)

57.toNull(component\_ob,str)

58.toNull(v\_delete,str)

59.toNull(component,str)

60.n = 0

61.конец условия

62.конец цикла

**DFS()**

1.vis[v] = 1;

2.component[n] = v;

3. для i=0 пока i<str делать i=i+1

4.если matrix[v][i] == 1 и vis[i] == 0

5.n = n + 1

6.вызвать функцию DFS(i,matrix,str,vis,component) для i

7.конец условия и цикла

Полный код программы можно увидеть в Приложении А.

# 4 Описание программы

Для написания данной программы использован язык программирования Java.Язык программирования Java - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.[4]

Проект был создан в виде консольного приложения Win32 (Visual C++).

Данная программа является многомодульной, поскольку состоит из нескольких функций: tmain, DFS, toNull, Connectivity, Component.

Работа программы начинается с запроса генерации матрицы. Если пользователь выбрал сгенерировать матрицу смежности, то на экран выводится запрос на количество вершин в орграфе. Затем выводятся запрос на ввод вероятности присутствия ребра между вершинами и возможность выбрать самостоятельного ввода вероятности, либо отказаться и автоматически программа будет формировать ребро с вероятностью 0,5.

System.*out*.println("Вы хотите сгенерировать матрицу смежности?(Y\\N)");  
Scanner in1 = new Scanner(System.*in*);  
var d = in1.next();  
if ((Objects.*equals*(d, "y")) || (Objects.*equals*(d, "Y"))){  
 System.*out*.println("Введите количество вершин в орграфе: ");  
 Scanner in2 = new Scanner(System.*in*);  
 *str* = in2.nextInt();  
  
 System.*out*.println("Вы хотите ввести вероятность генерации ребра?(Y\\N)");  
 Scanner in3 = new Scanner(System.*in*);  
 var pro = in3.next();  
 if ((Objects.*equals*(pro, "y")) || (Objects.*equals*(pro, "Y"))){  
 System.*out*.println("Введите вероятность генерации ребра от 0 до 1: ");  
 Scanner in4 = new Scanner(System.*in*);  
 *chance* = in4.nextDouble();  
 if (*chance* <= 1)  
 *chance* = *chance* \* 100;  
 }  
 else  
 System.*out*.println("Вероятность генерации ребра равна 0,5\n");

Далее выводятся элементы массива, а именно матрица смежности орграфа (рис.2).

*matrix* = new int[*str*][*str*];  
  
 System.*out*.println(*matrix*.length);  
 for (int i = 0; i < *str*; i++){  
 for (int j = 0; j < *str*; j++){  
 if (i == j)  
 *matrix*[i][j] = 0;  
 else{  
 int tmp = (int) (Math.*random*() \* 100);  
 if (tmp < *chance*)  
 *matrix*[i][j] = 1;  
 else  
 *matrix*[i][j] = 0;  
 }  
 }  
 }  
}

Если же пользователь отказался от случайной генерации, то ему предлагается выбрать один из трех орграфов разных видов, которые уже хранятся в памяти (рис.3).

if ((Objects.*equals*(d, "n")) || (Objects.*equals*(d, "N"))){  
 System.*out*.println("Выберите действие: ");  
 System.*out*.println("1.Вывести сильно связный орграф");  
 System.*out*.println("2.Вывести слабо связный орграф");  
 System.*out*.println("3.Вывести несвязный орграф");  
 System.*out*.println("Нажмите клавишу:");  
  
 Scanner in5 = new Scanner(System.*in*);  
 var action = in5.next();  
 *str* = 9;  
  
 *matrix* = new int[*str*][*str*];  
  
 if (Objects.*equals*(action, "1")){  
 for (int i = 0; i < *str*; i++)  
 for (int j = 0; j < *str*; j++)  
 *matrix*[i][j] = *StronglyCoupledDigraph*[i][j];  
 }  
 if (Objects.*equals*(action, "2")) {  
 for (int i = 0; i < *str*; i++)  
 for (int j = 0; j < *str*; j++)  
 *matrix*[i][j] = *WeaklyCoupledDigraph*[i][j];  
 }  
 if (Objects.*equals*(action, "3")) {  
 for (int i = 0; i < *str*; i++)  
 for (int j = 0; j < *str*; j++)  
 *matrix*[i][j] = *UnboundDigraph*[i][j];  
 }  
}

Для удобства пользователя все вершины пронумерованы и имеют свое уникальное название.

static int[][] *StronglyCoupledDigraph* = new int[][]{  
 {0,1,0,1,0,0,0,0,0},  
 {0,0,1,0,0,0,0,0,0},  
 {1,0,0,0,0,0,0,0,0},  
 {0,0,0,0,1,0,0,0,0},  
 {0,0,0,0,0,1,1,0,0},  
 {0,0,0,1,0,0,0,0,0},  
 {0,0,0,0,0,0,0,1,0},  
 {0,0,1,0,0,0,0,0,1},  
 {0,0,0,0,0,0,1,0,0}  
};  
static int[][] *WeaklyCoupledDigraph* = new int[][]{  
 {0,1,0,1,0,0,0,0,0},  
 {0,0,1,0,0,0,0,0,0},  
 {1,0,0,0,0,0,0,0,0},  
 {0,0,0,0,1,0,0,0,0},  
 {0,0,0,0,0,1,1,0,0},  
 {0,0,0,1,0,0,0,0,0},  
 {0,0,0,0,0,0,0,1,0},  
 {0,0,0,0,0,0,0,0,1},  
 {0,0,0,0,0,0,1,0,0}  
};  
static int[][] *UnboundDigraph* = new int[][]{  
 {0,1,0,0,0,0,0,0,0},  
 {0,0,1,0,0,0,0,0,0},  
 {1,0,0,0,0,0,0,0,0},  
 {0,0,0,0,1,0,0,0,0},  
 {0,0,0,0,0,1,1,0,0},  
 {0,0,0,1,0,0,0,0,0},  
 {0,0,0,0,0,0,0,1,0},  
 {0,0,0,1,0,0,0,0,1},  
 {0,0,0,0,0,0,1,0,0}  
};

Ниже можно увидеть оформление начального запроса и дальнейшие действия с ним.

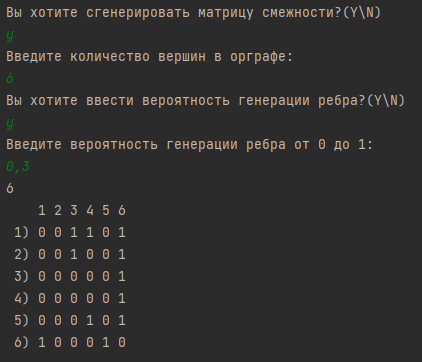


Рисунок 2 – Случайная генерация матрицы

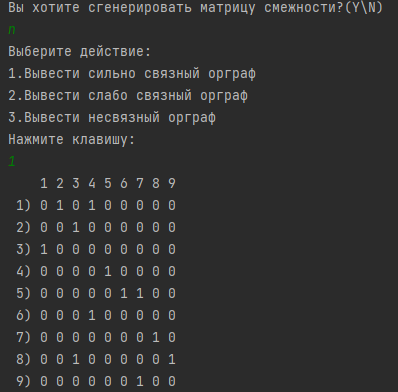


Рисунок 3 – Орграф выбран из памяти

Далее выводится вид орграфа и, если орграф связный, степень связности орграфа (рис. 4).

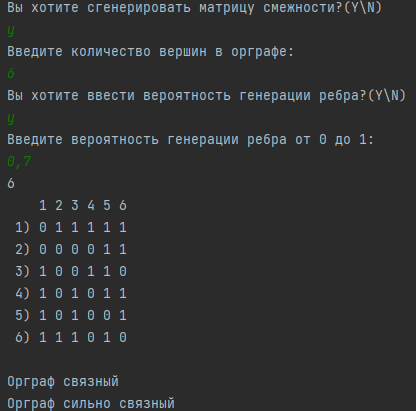


Рисунок 4 – Вид и степень связности орграфа

После происходит поиск компонент связности орграфа и вывод этих компонент, состоящих из вершин, на экран (рис. 5).

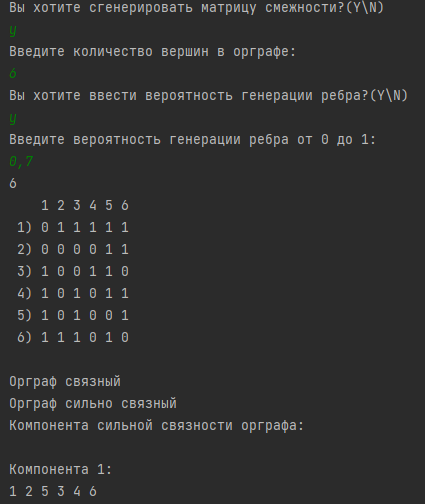


Рисунок 5 – Вывод компоненты сильной связности орграфа

# 5 Тестирование

Среда разработки IntelliJ IDEA Community Edition 2021.2.1 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин и вывод компонент сильной связности орграфов.

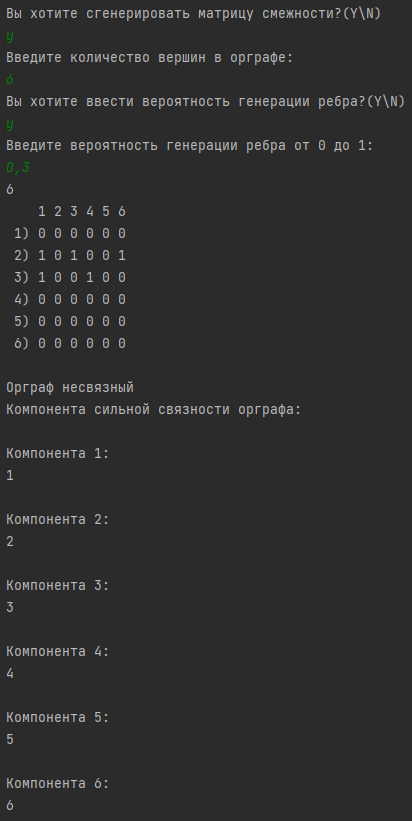


Рисунок 6 – Тестирование при вводе количество вершин = 6 и вероятностью = 0,3

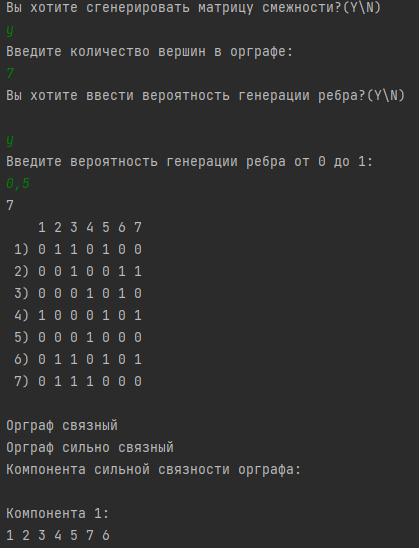


Рисунок 7 – Тестирование при вводе количество вершин = 7 и вероятностью = 0,5

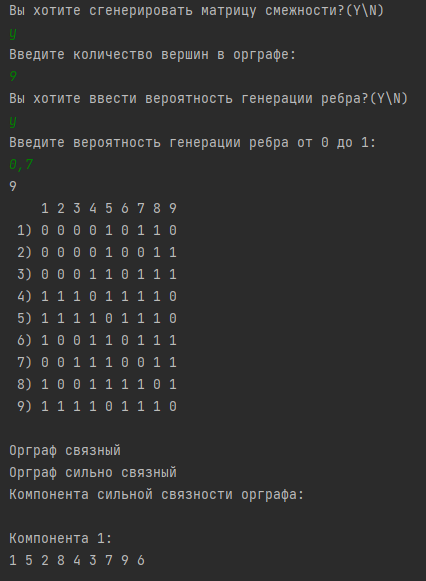


Рисунок 8 – Тестирование при вводе количество вершин = 9 и вероятностью = 0,7

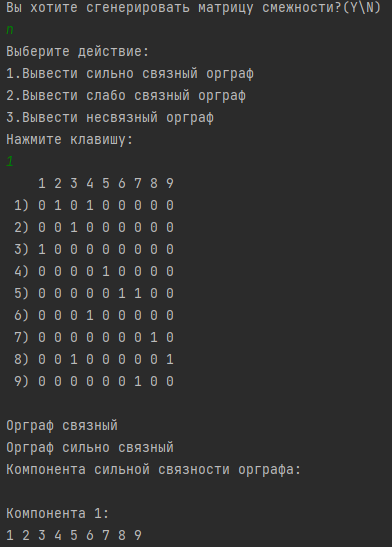


Рисунок 9– Тестирование при выводе сильно связного орграфа из памяти

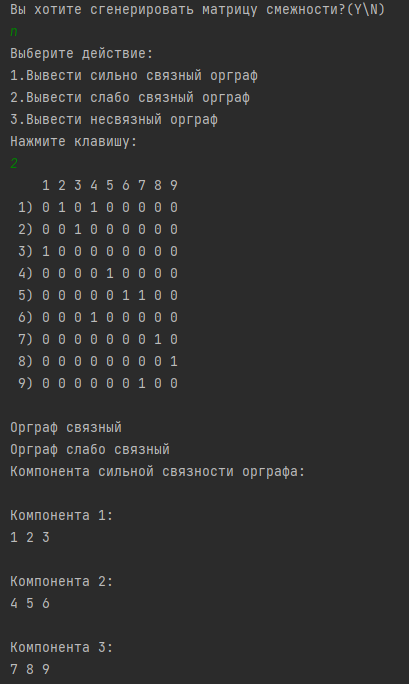


Рисунок 10 – Тестирование при выводе слабо связного орграфа из памяти

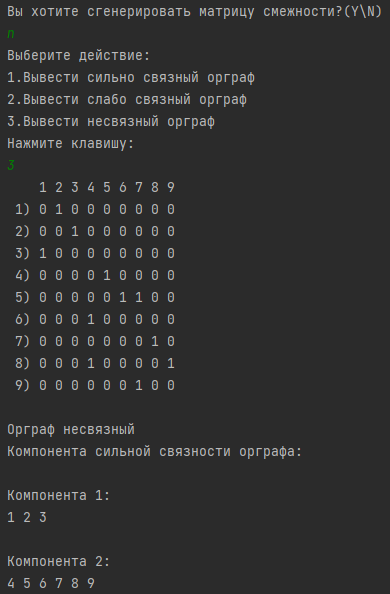


Рисунок 11 – Тестирование при выводе несвязного орграфа из памяти

В результате тестирования было выявлено, что программа успешно проверяет данные на соответствие необходимым требованиям.

# 6 Ручной расчёт задачи

Проведем проверку программы посредством ручных вычислений на примере графа с 9 вершинами (рисунок 6).

Начинаем обход из 0 вершины в последующие. Проверяем, если есть путь из 0 в другие вершины, то дальше идем. В нашем случае идем в вершину 1. Аналогично проверяем и в вершине 1. Из вершины 1 идем в 2. Из вершины 2 нет путей, которые нам не известны, поэтому возвращаемся в 1. Аналогично и в 1 вершине нет. Возвращаемся в 0 и проверяем. Из 0 есть путь в 3, идем туда. Из 3 идем в 4 вершину. Дальше в 5. Из вершины 5 нет путей нам не известных, поэтому возвращаемся в 4. Из 4 есть путь в 6, идем туда. Далее из 6 идем в 7 вершину. Из 7 вершины в 8. Из 8 вершины путей не пройденными нами нет. Поэтому возвращаемся в предыдущие и проверяем. Проверка показала, что все вершины пройдены. Мы выделяем компоненту связности {0 1 2 3 4 5 6 7 8}. И делаем вывод, что орграф связный.

Далее аналогичным способом мы запускаем обход из последующих вершин и сверяем получившиеся компоненты с компонентой вершины 0. В нашем случае компоненты вершин 1, 2 совпали с компонентой 0. Мы помечаем, что эти компоненты равны.

Начиная обход из вершины 3, мы нашли компоненту {3 4 5 6 7 8}. Она не схожа с компонентой вершины 0, мы помечаем эту вершину и делаем обход дальше. В нашем случае ни одна и последующих компонент вершин не совпала с компонентой вершины 0. Далее мы помеченные вершины удаляем из нашей компоненты, кроме вершин 1, 2, т.к. их компоненты совпали с компонентой 0. В итоге у нас получилась 1 компонента орграфа {0 1 2}.

Дальше мы берем следующую вершину, которая не входит в компоненту вершины 0, и делаем аналогичный обход.

* нашем случае мы начинаем с вершины 3. Делая обход, мы нашли компоненту этой вершины {3 4 5 6 7 8}.

Далее мы запускаем обход со следующих вершин. Получилось, что компоненты вершин 4, 5 совпали с компонентой вершины 3. Мы аналогично, как и с 0 вершиной, помечаем, что эти компоненты равны.

Обходя следующие вершины, мы обнаружили, что компоненты вершин 6, 7, 8 не совпадают с компонентой вершины 3. Мы их помечаем и когда обход закончен, также удаляем из компоненты вершины 3, кроме вершин 4,5, т.к. они равны. Мы нашли 2 компоненту орграфа {3 4 5}.

Дальше мы берем следующую вершину, которая не входит в компоненту вершины 3, и делаем аналогичный обход.

* + нашем случае мы начинаем с вершины 6. Делая обход, мы нашли компоненту этой вершины {6 7 8}. Обходя следующие вершины, мы обнаружили, что компоненты вершин 7, 8 совпадают с компонентой вершины

1. Следовательно, делаем вывод, что 3 компонента орграфа равна {6,7,8}.

В итоге у нас получилось 3 компоненты сильной связности орграфа {0 1 2}, {3 4 5}, {6 7 8}. Из этого делаем вывод, что наш граф не сильно связан.

Таким образом, можно сделать вывод, что программа работает верно.

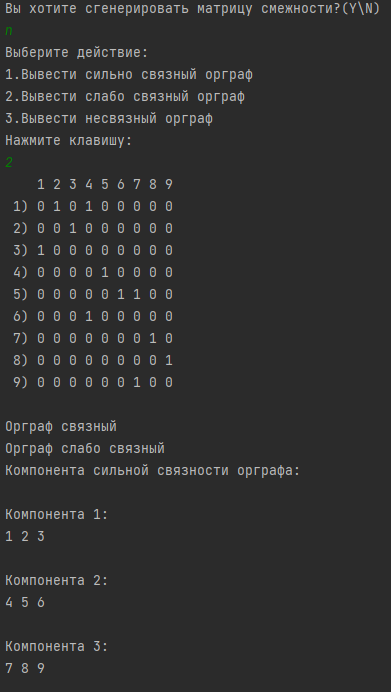


Рисунок 12 – Тестирование работы программы

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм поиска в глубину для поиска компонент сильной связности орграфа в IntelliJ IDEA Community Edition 2021.2.1.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, основанных на теории орграфов. Приобретены навыки по осуществлению алгоритма поиска в глубину. Углублены знания языка программированияCи.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Потому что программа работает в консольном режиме, не добавляющем к сложности языка сложность программного оконного интерфейса.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

# Список литературы

1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: Построение и анализ - М.: МЦНМО, 2001. - 960 с. URL:

<https://e-maxx.ru/bookz/files/cormen.pdf> (дата обращения 19.12.21)

1. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978. URL:

<https://studizba.com/files/show/pdf/53991-1-n-kristofides--teoriya-grafov.html> (дата обращения 19.12.21)

1. 3. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с. URL:

<https://obuchalka.org/20190715111515/grafi-i-ih-primenenie-ore-o-1965.html> (дата обращения 19.12.21)

1. Metanit Руководство по языку программирования Java. URL:

<https://metanit.com/java/tutorial/> (дата обращения 19.12.21)

# Приложение A Листинг программы.

import java.io.\*;  
import java.util.\*;  
  
public class Main {  
 static int count = 0;  
 static ArrayList<String> ComponentForOutput = new ArrayList<>();  
 static ArrayList<LinkedList<Integer>> VertexForOutput = new ArrayList<>();  
 static int str;  
 static int CountForOutput = 0;  
 static int componentCount = 0;  
 static int n = 0;  
 static int NumberOfAdjacentVertices = 0;  
 static int connectivityCounter = 0;  
 static int s = 0;  
 static int NumberOfComponent = 1;  
 static double chance = 50;  
 static int[][] matrix;  
 static int[] vis;  
 static int[] component;  
 static int[] component\_ob;  
 static int[] vDelete;  
 static int[] vSkip;  
  
 static int[][] StronglyCoupledDigraph = new int[][]{  
 {0,1,0,1,0,0,0,0,0},  
 {0,0,1,0,0,0,0,0,0},  
 {1,0,0,0,0,0,0,0,0},  
 {0,0,0,0,1,0,0,0,0},  
 {0,0,0,0,0,1,1,0,0},  
 {0,0,0,1,0,0,0,0,0},  
 {0,0,0,0,0,0,0,1,0},  
 {0,0,1,0,0,0,0,0,1},  
 {0,0,0,0,0,0,1,0,0}  
 };  
 static int[][] WeaklyCoupledDigraph = new int[][]{  
 {0,1,0,1,0,0,0,0,0},  
 {0,0,1,0,0,0,0,0,0},  
 {1,0,0,0,0,0,0,0,0},  
 {0,0,0,0,1,0,0,0,0},  
 {0,0,0,0,0,1,1,0,0},  
 {0,0,0,1,0,0,0,0,0},  
 {0,0,0,0,0,0,0,1,0},  
 {0,0,0,0,0,0,0,0,1},  
 {0,0,0,0,0,0,1,0,0}  
 };  
 static int[][] UnboundDigraph = new int[][]{  
 {0,1,0,0,0,0,0,0,0},  
 {0,0,1,0,0,0,0,0,0},  
 {1,0,0,0,0,0,0,0,0},  
 {0,0,0,0,1,0,0,0,0},  
 {0,0,0,0,0,1,1,0,0},  
 {0,0,0,1,0,0,0,0,0},  
 {0,0,0,0,0,0,0,1,0},  
 {0,0,0,1,0,0,0,0,1},  
 {0,0,0,0,0,0,1,0,0}  
 };  
  
 public static void DFS(int v, int[][] M, int str, int[] vis, int[] component) {  
 vis[v] = 1;  
 component[n] = v;  
  
 for (int i = 0; i < str; i++) {  
 if ((M[v][i] == 1) && (vis[i] == 0)) {  
 n++;  
 DFS(i, M, str, vis, component);  
 }  
 }  
 }  
  
 public static void toNull(int[] matr, int str) {  
 for (int i = 0; i < str; i++) {  
 matr[i] = 0;  
 }  
 }  
  
 public static int Connectivity(int[] vis, int str) {  
 var l = 0;  
 for (int i = 0; i < str; i++) {  
 if (vis[i] == 1)  
 l++;  
 }  
 if (l == str)  
 l = 1;  
 else  
 l = 0;  
 return l;  
 }  
  
 public static void Component(int[][] M, int str, int[] component, int[] v\_delete, int[] v\_propysk, int[] component\_ob, int[] vis) {  
 NumberOfAdjacentVertices = 0;  
 for (int i = 0; i < str; i++)  
 if (v\_propysk[i] == 0) {  
 DFS(i, M, str, vis, component);  
 int r = n;  
 componentCount = n;  
 s = 0;  
  
 for (int j = 0; j < componentCount + 1; j++)  
 component\_ob[j] = component[j];  
 toNull(vis, str);  
 toNull(component, str);  
 n = 0;  
 for (int j = 0; j < str; j++) {  
 DFS(j, M, str, vis, component);  
 for (int m = 0; m < componentCount + 1; m++) {  
 for (int t = 0; t < n + 1; t++)  
 if (component\_ob[m] == component[t])  
 NumberOfAdjacentVertices++;  
 }  
 if (n == componentCount) {  
 if (NumberOfAdjacentVertices == (componentCount + 1))  
 v\_propysk[j] = 1;  
 } else {  
 v\_delete[s] = j;  
 s++;  
 }  
  
 n = 0;  
 toNull(vis, str);  
 toNull(component, str);  
 NumberOfAdjacentVertices = 0;  
 }  
 r = ConnectivityCheck(v\_delete, component\_ob, r);  
 ComponentWrite(str, component, v\_delete, component\_ob, vis, r);  
 }  
 }  
  
 private static int ConnectivityCheck(int[] v\_delete, int[] component\_ob, int r) {  
 if (s != 0) {  
 for (int q = 0; q < s; q++) {  
 for (int z = 0; z < r + 1; z++)  
 if (component\_ob[z] == v\_delete[q]) {  
 component\_ob[z] = 0;  
 for (int x = z; x < r + 1; x++) {  
 if (x == r)  
 component\_ob[x] = 0;  
 else  
 component\_ob[x] = component\_ob[x + 1];  
 }  
 r--;  
 }  
 }  
 }  
 return r;  
 }  
  
 private static void ComponentWrite(int str, int[] component, int[] v\_delete, int[] component\_ob, int[] vis, int r) {  
 ComponentForOutput.add("Компонента " + NumberOfComponent + ":");  
 System.out.println(ComponentForOutput.get(count));  
 VertexForOutput.add(new LinkedList<>());  
 for (int y = 0; y < r + 1; y++) {  
 VertexForOutput.get(count).add(y, (component\_ob[y] + 1));  
 System.out.print(VertexForOutput.get(count).get(y) + " ");  
 }  
 count++;  
 System.out.println("\n");  
 NumberOfComponent++;  
 componentCount = 0;  
 toNull(vis, str);  
 toNull(component\_ob, str);  
 toNull(v\_delete, str);  
 toNull(component, str);  
 n = 0;  
 }  
  
 private static String baseMenu() {  
 System.out.println("Вы хотите сгенерировать матрицу смежности?(Y\\N)");  
 Scanner in1 = new Scanner(System.in);  
 var d = in1.next();  
 if ((Objects.equals(d, "y")) || (Objects.equals(d, "Y"))){  
 System.out.println("Введите количество вершин в орграфе: ");  
 Scanner in2 = new Scanner(System.in);  
 str = in2.nextInt();  
  
 System.out.println("Вы хотите ввести вероятность генерации ребра?(Y\\N)");  
 Scanner in3 = new Scanner(System.in);  
 var pro = in3.next();  
 if ((Objects.equals(pro, "y")) || (Objects.equals(pro, "Y"))){  
 System.out.println("Введите вероятность генерации ребра от 0 до 1: ");  
 Scanner in4 = new Scanner(System.in);  
 chance = in4.nextDouble();  
 if (chance <= 1)  
 chance = chance \* 100;  
 }  
 else  
 System.out.println("Вероятность генерации ребра равна 0,5\n");  
  
 matrix = new int[str][str];  
  
 System.out.println(matrix.length);  
 for (int i = 0; i < str; i++){  
 for (int j = 0; j < str; j++){  
 if (i == j)  
 matrix[i][j] = 0;  
 else{  
 int tmp = (int) (Math.random() \* 100);  
 if (tmp < chance)  
 matrix[i][j] = 1;  
 else  
 matrix[i][j] = 0;  
 }  
 }  
 }  
 }  
 return d;  
 }  
  
 private static void DeterminingTheDegreeOfConnectivityOfDigraph() {  
 CountForOutput = s;  
 if (s > 0){  
 System.out.println("Орграф связный");  
 DFS(0, matrix, str, vis, component);  
 for (int i = 0; i < n+1; i++)  
 component\_ob[i] = component[i];  
 componentCount = n;  
 toNull(vis, str);  
 toNull(component, str);  
 n = 0;  
 for (int m = 1; m < str; m++){  
 DFS(m, matrix, str, vis, component);  
 for (int i = 0; i < componentCount +1; i++){  
 for (int j = 0; j < n+1; j++)  
 if (component\_ob[i] == component[j])  
 NumberOfAdjacentVertices++;  
 }  
 if (NumberOfAdjacentVertices == str){  
 connectivityCounter++;  
 NumberOfAdjacentVertices = 0;  
 n = 0;  
 toNull(vis, str);  
 toNull(component, str);  
 }  
 }  
 if (connectivityCounter == (str - 1)){  
 System.out.println("Орграф сильно связный");  
 System.out.println("Компонента сильной связности орграфа: \n");  
 ComponentForOutput.add("Компонента " + NumberOfComponent + ":");  
 System.out.println(ComponentForOutput.get(count));  
 VertexForOutput.add(new LinkedList<>());  
 for (int i = 0; i < componentCount +1; i++){  
 VertexForOutput.get(count).add(i, (component\_ob[i] + 1));  
 System.out.print((VertexForOutput.get(count).get(i) + " "));  
 }  
 count++;  
 System.out.println();  
 }  
 else {  
 componentCount = 0;  
 n = 0;  
 toNull(vis, str);  
 toNull(component, str);  
 toNull(component\_ob, str);  
 System.out.println("Орграф слабо связный");  
 System.out.println("Компонента сильной связности орграфа: \n");  
 Component(matrix, str, component, vDelete, vSkip, component\_ob, vis);  
 }  
 }  
 else {  
 componentCount = 0;  
 toNull(component\_ob, str);  
 System.out.println("Орграф несвязный");  
 System.out.println("Компонента сильной связности орграфа: \n");  
 Component(matrix, str, component, vDelete, vSkip, component\_ob, vis);  
 }  
 }  
  
 private static void CountingTheNumberOfLinks() {  
 for (int i = 0; i < str; i++){  
 DFS(i, matrix, str, vis, component);  
 if (Connectivity(vis, str) == 1)  
 s++;  
 toNull(vis, str);  
 toNull(component, str);  
 n = 0;  
 }  
 toNull(vis, str);  
 toNull(component, str);  
 n = 0;  
 }  
  
 private static void InitializingArrays() {  
 vis = new int[str];  
 component = new int[str];  
 component\_ob = new int[str];  
 vDelete = new int[str];  
 vSkip = new int[str];  
  
  
 for (int i = 0; i < str; i++){  
 vis[i] = 0;  
 component[i] = 0;  
 component\_ob[i] = 0;  
 vDelete[i] = 0;  
 vSkip[i] = 0;  
 }  
 }  
  
 private static void Output() {  
 System.out.print(" ");  
 for (int i = 0; i < str; i++)  
 System.out.print(" " + (i + 1));  
 System.out.println();  
  
 for (int i = 0; i < str; i++){  
 System.out.print(" " + (i + 1) + ")");  
 for (int j = 0; j < str; j++){  
 System.out.print(" " + matrix[i][j]);  
 }  
 System.out.println();  
 }  
 System.out.println();  
 }  
  
 private static void mainMenu(String d) {  
 if ((Objects.equals(d, "n")) || (Objects.equals(d, "N"))){  
 System.out.println("Выберите действие: ");  
 System.out.println("1.Вывести сильно связный орграф");  
 System.out.println("2.Вывести слабо связный орграф");  
 System.out.println("3.Вывести несвязный орграф");  
 System.out.println("Нажмите клавишу:");  
  
 Scanner in5 = new Scanner(System.in);  
 var action = in5.next();  
 str = 9;  
  
 matrix = new int[str][str];  
  
 if (Objects.equals(action, "1")){  
 for (int i = 0; i < str; i++)  
 for (int j = 0; j < str; j++)  
 matrix[i][j] = StronglyCoupledDigraph[i][j];  
 }  
 if (Objects.equals(action, "2")) {  
 for (int i = 0; i < str; i++)  
 for (int j = 0; j < str; j++)  
 matrix[i][j] = WeaklyCoupledDigraph[i][j];  
 }  
 if (Objects.equals(action, "3")) {  
 for (int i = 0; i < str; i++)  
 for (int j = 0; j < str; j++)  
 matrix[i][j] = UnboundDigraph[i][j];  
 }  
 }  
 }  
  
 public static void WriteToFile(String filename, int[][] matrix, String d) throws IOException {  
 BufferedWriter outputWriter = null;  
 outputWriter = new BufferedWriter(new FileWriter(filename));  
  
 for (int i = 0; i < matrix.length; i++) {  
 outputWriter.write(Arrays.toString(matrix[i]));  
 outputWriter.newLine();  
 }  
 outputWriter.newLine();  
 outputWriter.newLine();  
  
 if (CountForOutput > 0) {  
 outputWriter.write("Орграф связный");  
 if (connectivityCounter == (str - 1)) {  
 outputWriter.newLine();  
 outputWriter.write("Орграф сильно связный");  
 }  
 else {  
 outputWriter.newLine();  
 outputWriter.write("Орграф слабо связный");  
 }  
 }  
 else {  
 outputWriter.write("Орграф несвязный");  
 }  
 outputWriter.newLine();  
 outputWriter.write("Компонента сильной связности орграфа:");  
  
 for (int i = 0; i < count; i++){  
 outputWriter.newLine();  
 outputWriter.newLine();  
 outputWriter.write(ComponentForOutput.get(i));  
 outputWriter.newLine();  
 for (int j = 0; j < VertexForOutput.get(i).size(); j++){  
 outputWriter.write(VertexForOutput.get(i).get(j) + " ");  
 }  
 }  
  
 outputWriter.flush();  
 outputWriter.close();  
 }  
  
 public static void main(String[] args) throws IOException {  
 String d = baseMenu();  
 mainMenu(d);  
 Output();  
 InitializingArrays();  
 CountingTheNumberOfLinks();  
 DeterminingTheDegreeOfConnectivityOfDigraph();  
  
 String filename = "test";  
 WriteToFile(filename, matrix, d);  
 }  
}