Министерство образования Российской федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**Пояснительная записка**

К курсовому проектированию по курсу «Логика и основы алгоритмизации в инженерных задачах» на тему «Реализация унарных и бинарных операций в графах»

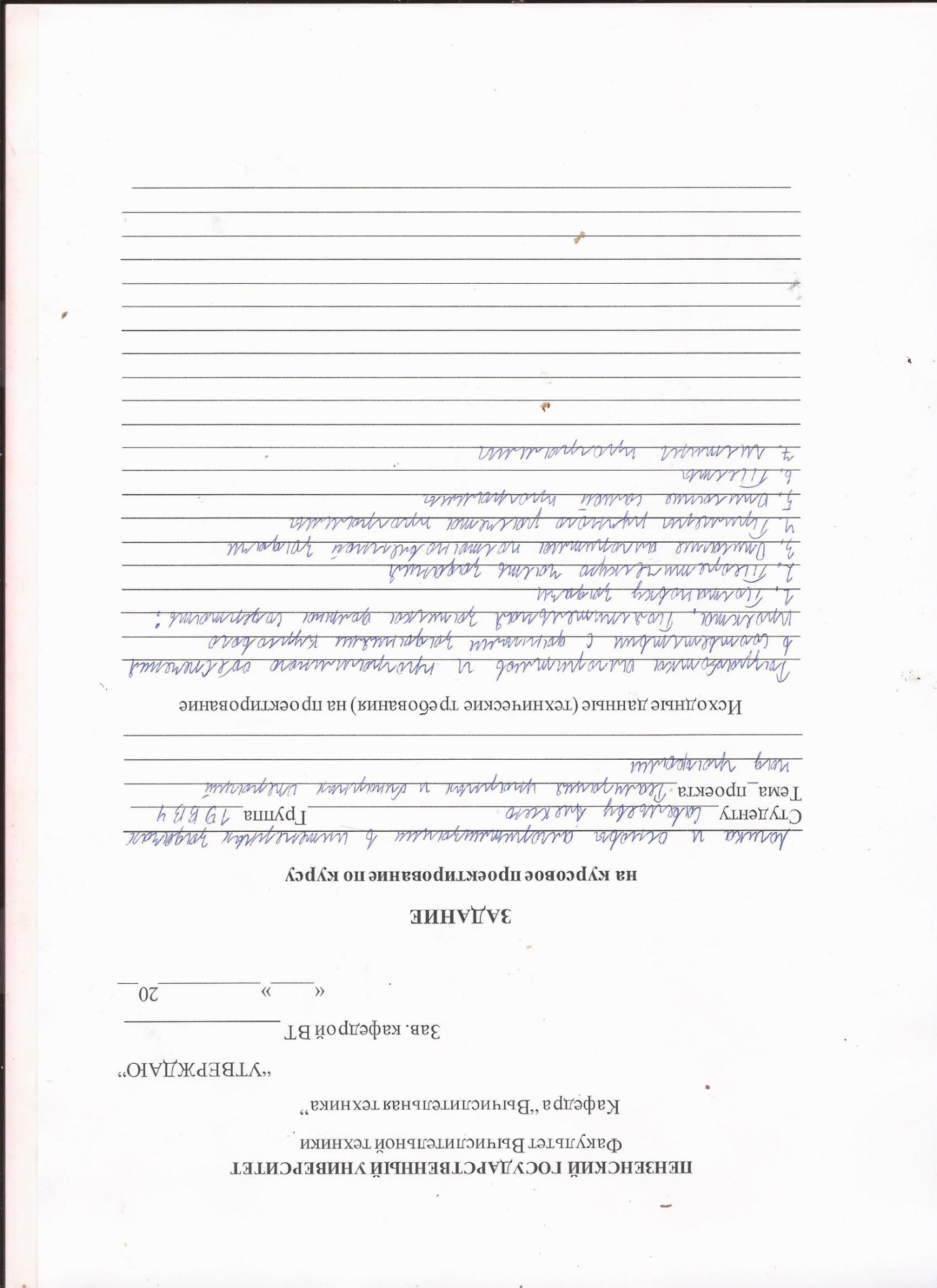
Выполнил студент группы 19ВВ4:

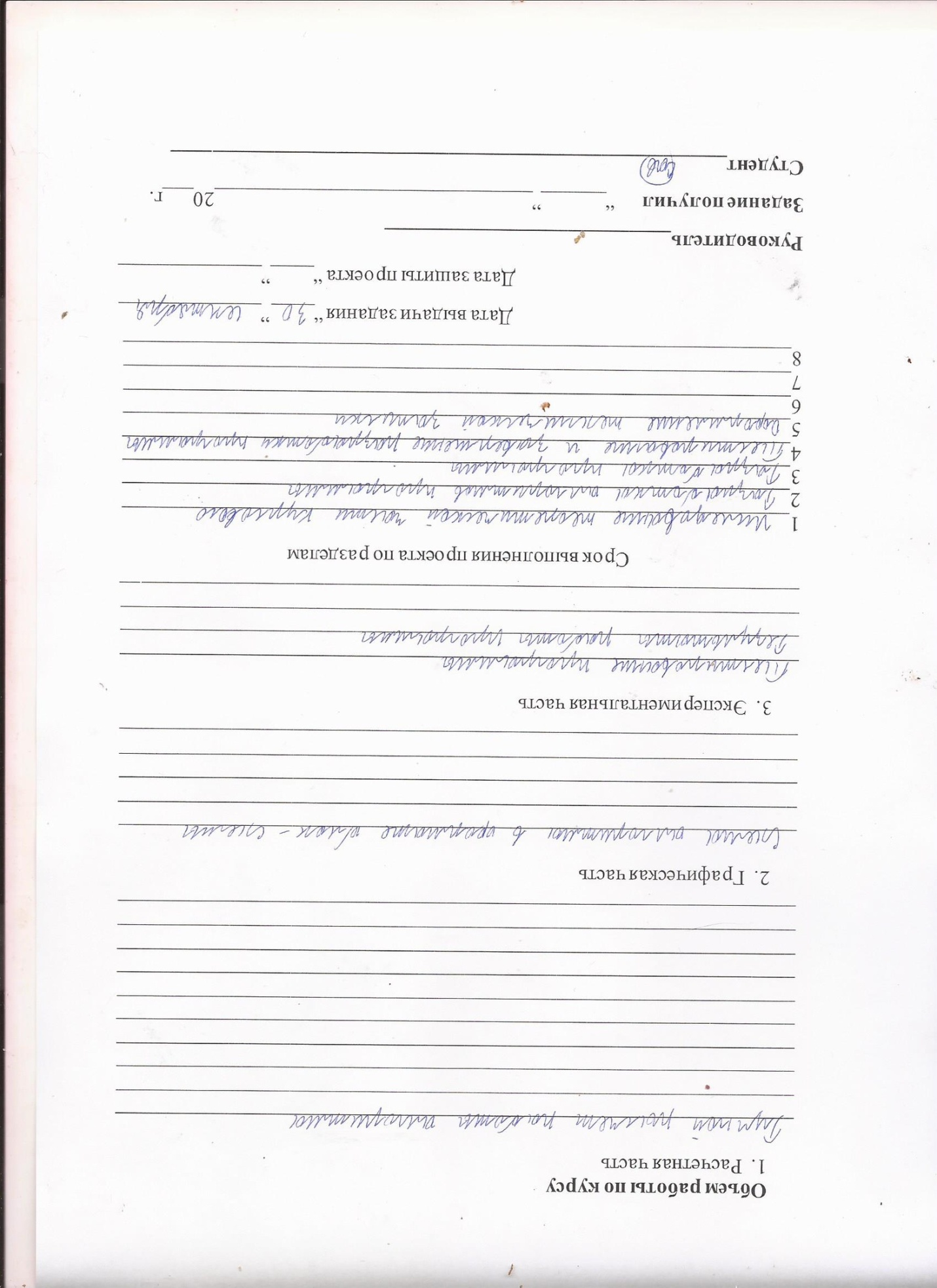
Савельев А.А.

Принял:

Митрохин М.А.

Пенза 2021

****

**Содержание:**

**Реферат…………………………………………………………………………………………………………5**

**Введение……………………………………………………………………………………………………….6**

**1. Постановка задачи……………………………………………………………………………………7**

**2. Теоретическая часть задания…………………………………………………………………..8**

**3. Описание алгоритмов……………………………………..……………………………………..11**

**4. Описание программы……………………………………………………………………………..25**

**5. Тестирование…………………………………………………………………………………………..26**

**6. Ручной расчет………………………………………………………………………………………….28**

**Заключение……………………………………………………………………………….....................38**

**Приложение A.………………………………..………………………………..………………………….39**

***Реферат***

**59 стр, 21 рис.**

Унарные и Бинарные операции

Цель исследования – разработка программы, способная выполнять унарные и бинарные функции над графом.

В работе рассмотрены правила выполнения унарных и бинарных операций и принципы их работы на матрицах смежности на примере операций: отождествление вершин, стягивание ребра, расщепление вершины, пересечение графов, объединение графов, кольцевая сумма графов, декартово произведение графов.

***Введение***

**Операции над графами** образуют новые графы из старых. Операции можно разделить на следующие основные категории. Унарные операции - одноместная операция создаёт новый граф из старого. Бинарные операции - двуместная операция создаёт новый граф из двух исходных графов *G1(V1, E1)* и *G2(V2, E2)*.

В качестве среды разработки мною была выбрана среда MicrosoftVisualStudio2019, язык программирования – Си++.

Целью данной курсовой работы является разработка программы на языке Си++, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуются операции над графами.

***1. Постановка задачи***

Требуется разработать программу, которая позволит выполнять над графами следующие бинарные и унарные операции:

* Отождествление вершин
* Стягивание ребра
* Расщепление вершин
* Пересечение графов
* Объединение графов
* Кольцевая сумма графов
* Декартово произведение

Исходный граф в программе должен задаваться матрицей смежности, причѐм при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матриц смежности. Устройство ввода – клавиатура и мышь.

***2. Теоретическая часть задания***

***Отождествление вершин****.* В графе *G*1 выделяются вершины *и,v.* Определяют окружение *Q*1 вершины *u*,и окружение *Q*2 вершины *v,* вычисляют их объединение *Q* = *Q1* * Q2.* Затем над графом *G*1 выполняются следующие преобразования:

* из графа *G*1 удаляют вершины *u,* *v (H*1 *= G*1 *- u - v);*
* к графу *Н*1присоединяют новую вершину *z (H*1 *= H*1 *+z);*
* вершину *z* соединяют ребром с каждой из вершин *w*1*Q*

*(G*2 *= H*1 *+ zwi*, *i =* 1,2,3*,…).*

***Стягивание ребра.*** Данная операция является операцией отождествления смежных вершин *и, v* в графе *G*1.

***Расщепление вершины.*** В графе G1 выделяется вершина ν. Окружение Q вершины ν разбивается на две части М, N (Mhttps://cde.osu.ru/courses2/temp4/images/und.gifN=Q, Mhttps://cde.osu.ru/courses2/temp4/images/sh.gifN=ø). После этого над графом G1 выполняются следующие преобразования:

* удаляют вершину ν вместе с инцидентными ей ребрами (Н1=G1=ν-νzi, zihttps://cde.osu.ru/courses2/temp4/images/in.gifQ, i=1,2,3 ..);
* добавляют новые вершины u,w и соединяющее их ребро uw(H2=H1+u+w+uw);
* вершину u соединяют ребром с каждой вершиной из множества М(H3=H1+uzi, zihttps://cde.osu.ru/courses2/temp4/images/in.gifM);
* вершину w соединяют ребром с каждой вершиной из множества N(G2=H3+uzi, zihttps://cde.osu.ru/courses2/temp4/images/in.gifM).

***Объединение.*** Граф *G* называется объединением или наложением графов *G*1 и *G*2, если *VG = V*1*V*2*; UG = U*1* U*2 (рис. 1).

**U**

*v*1

*v*2

*v*3

*v*4

*v*3

*v*4

*v*5

*v*2

*v*1

*v*3

*v*4

*v*5

Рисунок 1 — Объединение графов *G*1, *G*2

Объединение графов *G*1 и *G*2 называется дизъюнктным, если *V*1*V*2 *= *. При дизъюнктном объединении никакие два из объединяемых графов не должны иметь общих вершин.

***Пересечение.*** Граф *G* называется пересечением графов *G*1, *G*2,если *VG = V*1*V*2и *UG = U*1*U*2 (риc.2). Операция "пересечения" записывается следующим образом: *G = G*1*G*2*.*

**∩**

*v*1

*v*2

*v*3

*v*5

*v*3

*v*4

*v*6

*v*2

*v*1

*v*6

*v*4

*v*5

*v*1

*v*4

*v*6

*v*5

*v*3

*v*2

Рисунок 2 — Пересечение графов *G*1, *G*2*.*

***Кольцевая сумма*** графов представляет граф, который не имеет изолированных вершин и состоит из ребер, присутствующих либо в первом исходном графе, либо во втором. Кольцевая сумма определяется следующим соотношением: *G* = *G*1  *G*2 (рис.4).

**⊕**

*v*1

*v*2

*v*3

*v*5

*v*3

*v*4

*v*6

*v*2

*v*1

*v*4

*v*5

*v*1

*v*4

*v*6

*v*5

*v*3

*v*2

Рисунок 3 — Кольцевая сумма графов *G1, G2*

***Декартово произведение.***Граф *G* называется декартовым произведением графов *G*1 и *G*2 если *VG* = *V*1*V*2 —декартово произведение множеств вершин графов *G*1, *G*2, а множество ребер *U*c задается следующим образом: вершины (*zi*, *vk*) и (*zj*, *vl*) смежны в графе *G* тогда и только тогда, когда *zi* = *zj*(*i* = *j*), a *v*k и *vl* смежны в *G*2 или *vk* = *vl*(*k* = *l*), смежны в графе *G*1 (см. рис.3).

**X**

*z*1

*z*2

*v*1

*v*3

*v*2

*z*1*v*1

*z*1*v*2

*z*1*v*3

*z*2*v*1

*z*2*v*2

*z*2*v*3

Рисунок 4 — Декартово произведение графов *G*1, *G*2

***3. Описание алгоритмов программы***

**Алгоритм отождествления вершин**

Для отождествления поэлементно логически складываются строки матрицы под номерами выбранных вершин.

Полученный результат сложения записывается в одну из этих строчек, а другая строка, как и вершина удаляется. В результирующей матрице, ребро между этими вершинами становится петлей.

Псевдокод функции Otoshdestvlenie:

**Otoshdestvlenie()**

1. для i=0 пока i<n делать i=i+1

2. mas[v2][j] = (mas[v1][j] + mas[v2][j]);

3. Если mas[v2][j] == 2 то mas[v2][j] = 1;

4. Конец цикла

5. Если v2 меньше или равно n / 2 то

6. для i=0 пока i<n делать i=i+1

7. для j=0 пока j<n делать j=j+1

8. mas[j][i] = mas[i][j];

9. Если i == j то mas[i][j] = 0;

10. Если i == j && i == v2 то mas[i][j] = 1;

11. Конец цикла

12.Конец цикла

13. Иначе

14. Если v2 меньше или равно n / 2 то

15. для i= n - 1 пока i>=0 делать i=i-1

16. для j= n - 1 пока j<0 делать j=j-1

17. mas[j][i] = mas[i][j];

18. Если i == j то mas[i][j] = 0;

19. Если i == j && i == v2 то mas[i][j] = 1;

20. Конец цикла

21.Конец цикла

22. для i=0 пока i<n делать i=i+1

23. для j=0 пока j<n делать j=j+1

24. если i == v1 то прервать цикл

25. если j == v1 то перейти к следующей итерации цикла

26. massa[i][j] = mas[i][j];

27. Конец цикла

28. Конец цикла

29. для i=0 пока i<n делать i=i+1

30. для j=0 пока j<n делать j=j+1

31. если k >= (n - 1)

32. l = l + 1

33. k = 0

34. если massa[i][j] >= 0

35. massa2[l][k] = massa[i][j];

36. k= k + 1

37. Конец цикла

38. Конец цикла

39. для i=0 пока i<n делать i=i+1

40. для j=0 пока j<n делать j=j+1

41. вывод massa2

42. Конец цикла

43. Конец цикла

**Алгоритм стягивание ребра**

Аналогичен алгоритму отождествления, только имеет ограничение в выборе вершин. Вершины должны быть смежны. В результирующей матрице, ребро между этими вершинами НЕ становится петлей.

Псевдокод функции Styagivanie:

**Styagivanie()**

1. для i=0 пока i<n делать i=i+1

2. mas[v2][j] = (mas[v1][j] + mas[v2][j]);

3. Если mas[v2][j] == 2 то mas[v2][j] = 1;

4. Конец цикла

5. Если v2 меньше или равно n / 2 то

6. для i=0 пока i<n делать i=i+1

7. для j=0 пока j<n делать j=j+1

8. mas[j][i] = mas[i][j];

9. Если i == j то mas[i][j] = 0;

10. Конец цикла

11.Конец цикла

12. Иначе

13. Если v2 меньше или равно n / 2 то

14. для i= n - 1 пока i>=0 делать i=i-1

15. для j= n - 1 пока j<0 делать j=j-1

16. mas[j][i] = mas[i][j];

17. Если i == j то mas[i][j] = 0;

18. Конец цикла

19.Конец цикла

20. для i=0 пока i<n делать i=i+1

21. для j=0 пока j<n делать j=j+1

22. если i == v1 то прервать цикл

23. если j == v1 то перейти к следующей итерации цикла

24. massa[i][j] = mas[i][j];

25. Конец цикла

26. Конец цикла

27. для i=0 пока i<n делать i=i+1

28. для j=0 пока j<n делать j=j+1

29. если k >= (n - 1)

30. l = l + 1

31. k = 0

32. если massa[i][j] >= 0

33. massa2[l][k] = massa[i][j];

34. k= k + 1

35. Конец цикла

36. Конец цикла

37. для i=0 пока i<n делать i=i+1

38. для j=0 пока j<n делать j=j+1

39. вывод massa2

40. Конец цикла

41. Конец цикла

**Алгоритм расщепления вершины**

Сначала создается два множества вершин. Путем разделения на четные и нечетные вершины, создается две строки из одной. Далее вторая строка встает на место после первой строки. С помощью транспонирования мы точно также вставляем строку(столбец), чтобы получить квадратную матрицу.

Для транспонирования используется массив massaTrans. Для сохранения строк используются динамические одномерные массивы-строки.

Псевдокод функции Rascheplenie:

**Rascheplenie()**

1. для j=0 пока j<m делать j=j+1

2. Если j % 2 == 1

3. tempmas2[j] = mas2[Rv][j];

4. mas2[Rv][j] = 0;

5. mas2[j][Rv] = 0;

6. Если j % 2 == 0

7. tempmas2[j] = 0;

8. Если j == Rv

9. tempmas2[j] = 1;

10. Конец цикла

11. для i=0 пока i<m + 1 делать i=i+1

12. для j=0 пока j<m + 1 делать j=j+1

13. massa[i][j] = mas2[i][j]

14. Если i == Rv

15. massa[i][j] = tempmas2[j];

16. Если i == Rv

17. для k=0 пока k>i делать k=k-1

18. mas2[k] = mas2[k - 1]

19. Конец цикла

21. Конец цикла

22. Конец цикла

23. для i=0 пока i<m + 1 делать i=i+1

24. для j=0 пока j<m + 1 делать j=j+1

25. massaTrans[i][j] = massa[j][i]

26. Конец цикла

27. Конец цикла

28. для j=0 пока j<m + 1 делать j=j+1

29. tempmas2[j] = massa[Rv][j]

30. Конец цикла

31. для i=0 пока i<m + 1 делать i=i+1

32. tempmas22[i] = massaTrans[Rv][i];

33. Конец цикла

34. для i=0 пока i<m + 1 делать i=i+1

35. для j=0 пока j<m + 1 делать j=j+1

36. Если i == Rv

37. massaTrans[i][j] = tempmas2[j];

38. Если i == Rv

39. для k=0 пока k>i делать k=k-1

40. massaTrans[k] = massaTrans[k - 1]

41. Конец цикла

42. Конец цикла

43. Конец цикла

44. для i=0 пока i<m + 1 делать i=i+1

45. massaTrans[Rv][i] = tempmas22[i]

46. Конец цикла

47. для i=0 пока i<m + 1 делать i=i+1

48. для j=0 пока j<m + 1 делать j=j+1

49. massa[i][j] = massaTrans[j][i]

50. Конец цикла

51. Конец цикла

52. для i=0 пока i<m + 1 делать i=i+1

53. для j=0 пока j<m + 1 делать j=j+1

54. Если i == j

55. massa[i][j] = 0

56. massa[j][Rv] = massa[Rv][j];

57 Вывод massa[i][j]

58. Конец цикла

59. Конец цикла

**Алгоритм пересечения графов**

Поэлементно сравниваются две матрицы смежности. В результате остаются только те ребра, которые имеются в обоих матрицах одновременно.

Псевдокод функции Peresechenie:

**Peresechenie()**

1. для i=0 пока i< bol делать i=i+1

2. для j=0 пока j< bol делать j=j+1

**3.** massa[i][j] = mas2[i][j];

4. Конец цикла

5. Конец цикла

6. для i=0 пока i< men делать i=i+1

7. для j=0 пока j< men делать j=j+1

8. Если mas[i][j] == 1 и mas2[i][j] == 1

9. massa[i][j] = 1;

10. Иначе

11. massa[i][j] = 0;

12. Конец цикла

13. Конец цикла

14. для i=0 пока i< men делать i=i+1

15. для j=0 пока j< men делать j=j+1

16. Вывод massa[i][j]

17. Конец цикла

18. Конец цикл

**Алгоритм объединения графов**

Поэлементно сравниваются две матрицы смежности. В результате остаются ребра, которые имеются в любой из двух матриц.

Псевдокод функции obedenenie:

**obedenenie()**

1. для i=0 пока i< bol делать i=i+1

2. для j=0 пока j< bol делать j=j+1

**3.** massa[i][j] = mas2[i][j];

4. Конец цикла

5. Конец цикла

6. для i=0 пока i< men делать i=i+1

7. для j=0 пока j< men делать j=j+1

8. Если mas[i][j] == 1 или mas2[i][j] == 1

9. massa[i][j] = 1;

10. Иначе

11. massa[i][j] = 0;

12. Конец цикла

13. Конец цикла

14. для i=0 пока i< men делать i=i+1

15. для j=0 пока j< men делать j=j+1

16. Вывод massa[i][j]

17. Конец цикла

18. Конец цикла

**Алгоритм кольцевой суммы графов**

Поэлементно сравниваются две матрицы смежности. В результате остаются ребра, которые имеются в любой из двух матриц, но не в двух одновременно.

Псевдокод функции Kolcevaya\_summa:

**Kolcevaya\_summa**

1. для i=0 пока i< bol делать i=i+1

2. для j=0 пока j< bol делать j=j+1

**3.** massa[i][j] = mas2[i][j];

4. Конец цикла

5. Конец цикла

6. для i=0 пока i< men делать i=i+1

7. для j=0 пока j< men делать j=j+1

8. Если mas[i][j] == 1 или mas2[i][j] == 1

**9.** massa[i][j] = 1;

10. Если mas[i][j] == 1 и mas2[i][j] == 1

11. massa[i][j] = 0;

12. Если mas[i][j] == 0 и mas2[i][j] == 0

13. massa[i][j] = 0;

14. Конец цикла

15. Конец цикла

16. для i=0 пока i< men делать i=i+1

17. для j=0 пока j< men делать j=j+1

18. Вывод massa[i][j]

19. Конец цикла

20. Конец цикла

**Алгоритм декартова произведения**

Умножаем количество вершин каждой матрицы. Выбираем из двух матриц б**о**льшую. Копируем ее столько раз, сколько вершин в наименьшей матрице.

Далее проверяем на смежность элементы наименьшей матрицы.

Если элементы смежные, то добавляем ребра между скопированными матрицами.

Псевдокод функции decart:

**decart()**

1. Если bol == men

2. для i=0 пока i < bol делать i=i+1

3 . для j=0 пока j < bol делать j=j+1

4. Если mas1[i][j] == 1

5. ch1 = ch1 + 1

6. Если mas2[i][j] == 1 ch2 = ch2 + 1

7. Конец цикла

8. Конец цикла

9. Если ch1 < ch2

10. для i=0 пока i < bol делать i=i+1

11. для j=0 пока j < bol делать j=j+1

12. mastemp[i][j] = mas1[i][j];

13. mas1[i][j] = mas2[i][j];

14. mas2[i][j] = mastemp[i][j];

15. Конец цикла

16. Конец цикла

17. для i=0 пока i < bol \* men делать i=i+1

18. для j=0 пока j < bol \* men делать j=j+1

19. result[i][j] = 0;

20. Конец цикла

21. Конец цикла

22 . для k=0 пока k < men делать k=k+1

23. для i=0 пока i < bol делать i=i+1

24. для j=0 пока j < bol делать j=j+1

25. result[i][j] = mas1[z][x];

26. Конец цикла

27. Конец цикла

28. i = 0;

29. j = 0;

30. mn = mn + bol;

31. Конец цикла

32. для i = 0 пока i < men делать i++, verI = verI + bol, verJ = verJ + bol

33. для j=0 пока j < men делать j=j+1

34. Если mas2[i][j] == 1

35. mas2[j][i] = 0

36. mnoj = (bol \* j) - (i \* bol)

37. для с=0 пока с < bol делать с=с+1

38. result[verI + c][verJ + mnoj + c] = 1;

39. Конец цикла

40. Конец цикла

41. mnoj = 0;

42.Конец цикла

43. для i=0 пока i < bol \* men делать i=i+1

44. для j=0 пока j < bol \* men делать j=j+1

45. result[j][i] = result[i][j];

46. Конец цикла

47. Конец цикла

48. для i=0 пока i < bol \* men делать i=i+1

49. для j=0 пока j < bol \* men делать j=j+1

50. Вывод result[i][j]

***4. Описание программы***

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

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

Данная программа является многомодульной, поскольку состоит из нескольких функций: Otoshdestvlenie; Styagivanie; Rascheplenie; Peresechenie;

obedenenie; Kolcevaya\_summa

Работа программы начинается с запроса размеров матриц mas1 и mas2. Для каждой операции выделяется отдельная память для сохранения исходной матрицы massave и для матрицы результата massa. Все матрицы генерируются квадратными. Они выводятся на экран.

После чего каждая операция производится по очереди и на экран выводится результат каждой операции.

***5. Тестирование***

Среда разработки Microsoft Visual Studio 2010 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.

|  |  |  |
| --- | --- | --- |
| **Описание теста** | **Ожидаемый результат** | **Полученный результат** |
| Запуск программы | Вывод сообщения о выборе размеров матриц смежности | Верно |
| Выбор операции | Вывод сообщения о выборе операции под номерами | Верно |
| Работа операции  отождествления вершин | Вывод на экран верно обработанной матрицы смежности, имеющей на одну вершину меньше, а так же вершину с петлей | Верно |
| Работа операции  стягивания ребра | Вывод на экран верно обработанной матрицы смежности, имеющей на одну вершину меньше | Верно |
| Работа операции  расщепления вершин | Вывод на экран верно обработанной матрицы смежности, имеющей на одну вершину больше | Верно |
| Работа операции  пересечения графов | Вывод на экран верно обработанной матрицы смежности, полученной из двух матриц | Верно |
| Работа операции  объединения графов | Вывод на экран верно обработанной матрицы смежности, полученной из двух матриц смежности | Верно |
| Работа операции  кольцевой суммы | Вывод на экран верно обработанной матрицы смежности, полученной из двух матриц смежности | Верно |
| Работа операции  декартова произведения | Вывод на экран верно обработанной матрицы смежности, полученной из двух матриц смежности | Верно |

***6. Ручной расчѐт задачи***

**Расчет функции отождествления:**

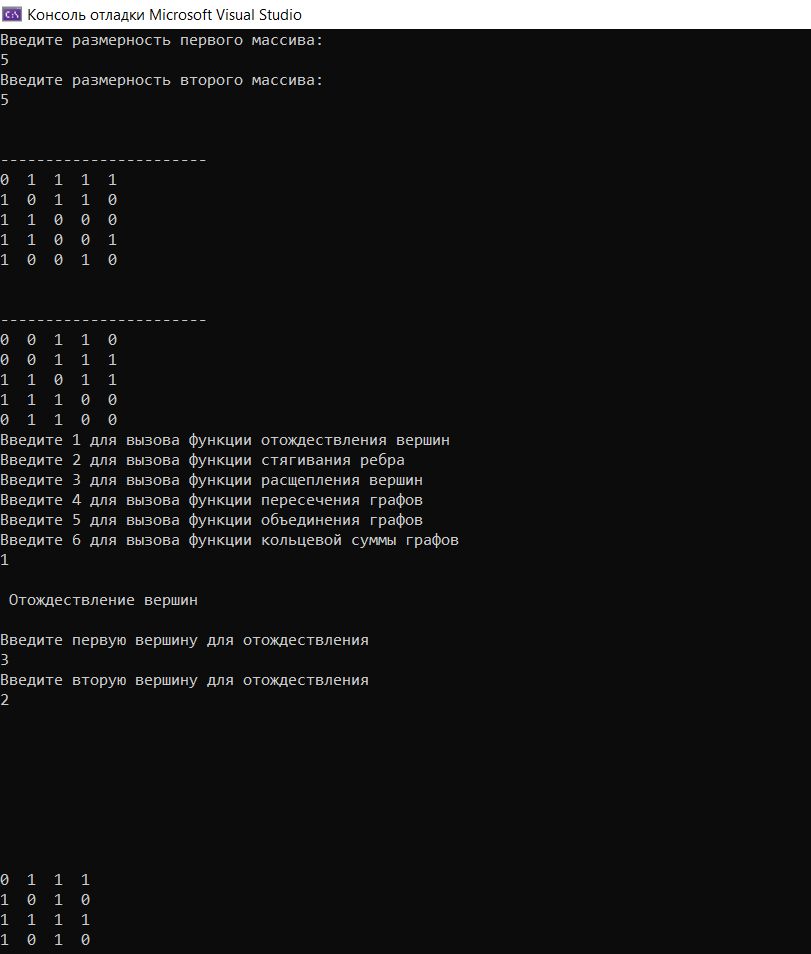


Рисунок 5 — Работа функции отождествления

Данная функция работает над первым массивом. Удаляется вторая введенная вершина, в нашем случае вершина номер 2

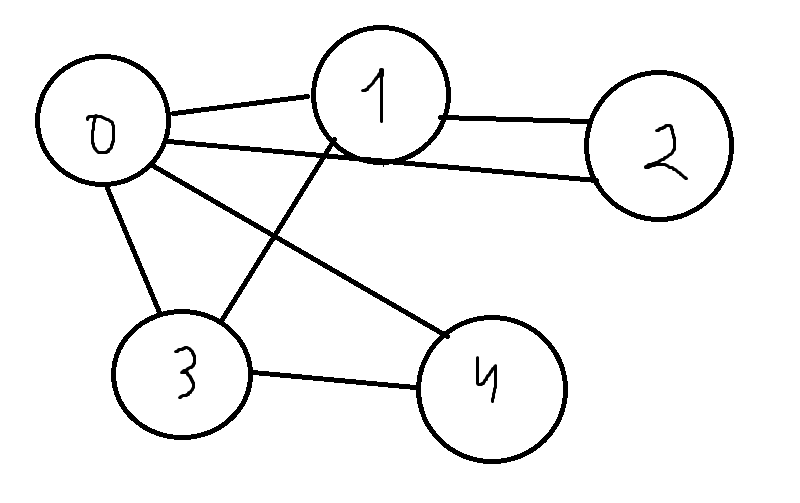


Рисунок 6 — Исходный граф

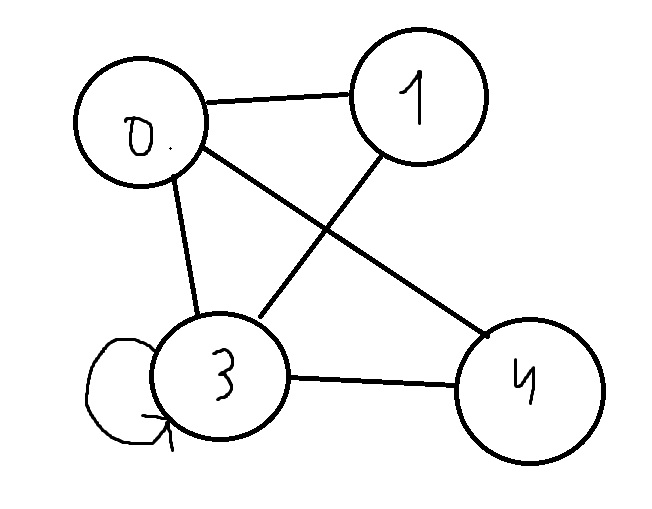


Рисунок 7 — Граф после преобразований

**Расчет функции стягивания ребра:**

Данная функция работает над первым массивом.

Удаляется вторая введенная вершина, в нашем случае вершина номер 2

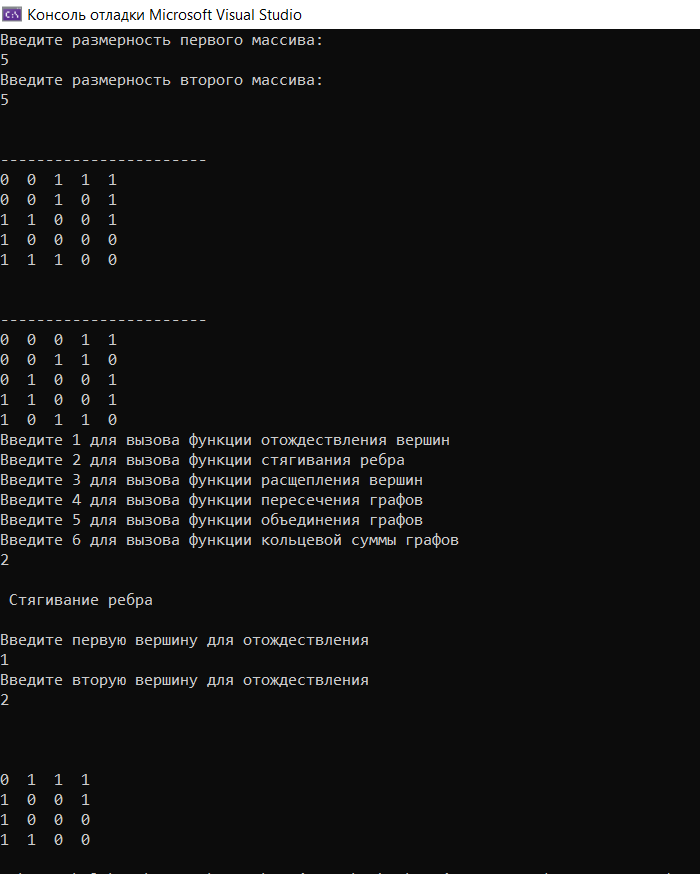


Рисунок 8 — Выполнение функции стягивания ребра

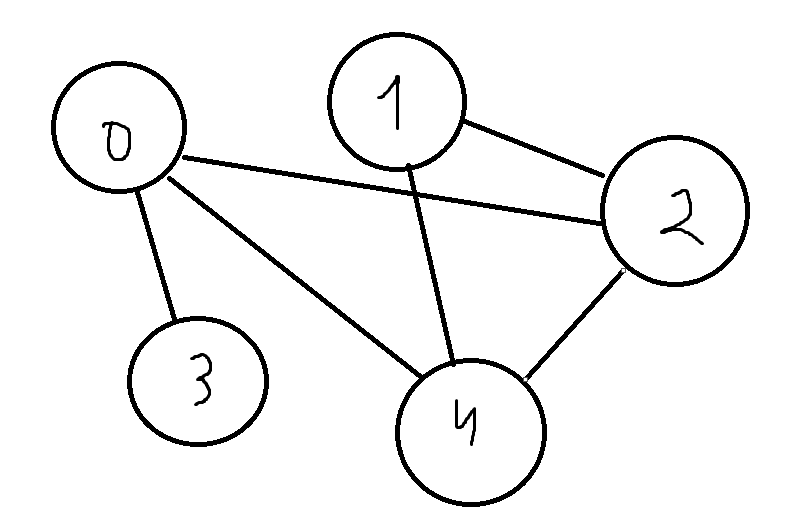


Рисунок 9 — Исходный граф

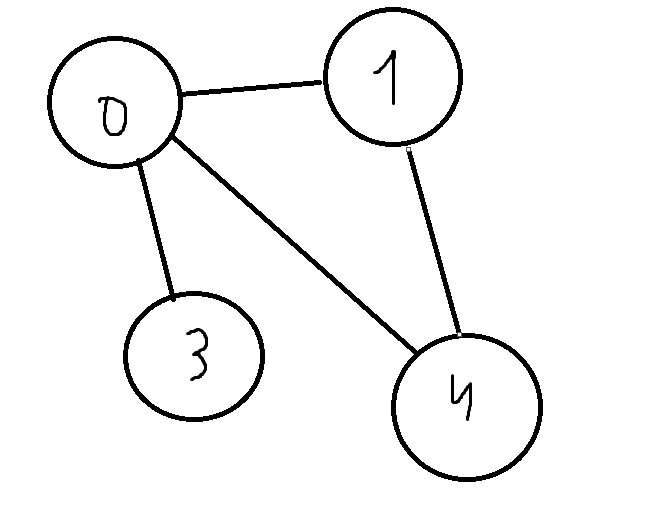


Рисунок 10 — Граф после преобразований

**Расчет функции расщепления:**

Данная функция работает над первым массивом. Выбранная вершина разделяется на две по четным и нечетным элементам. В нашем случае это вершина 0.

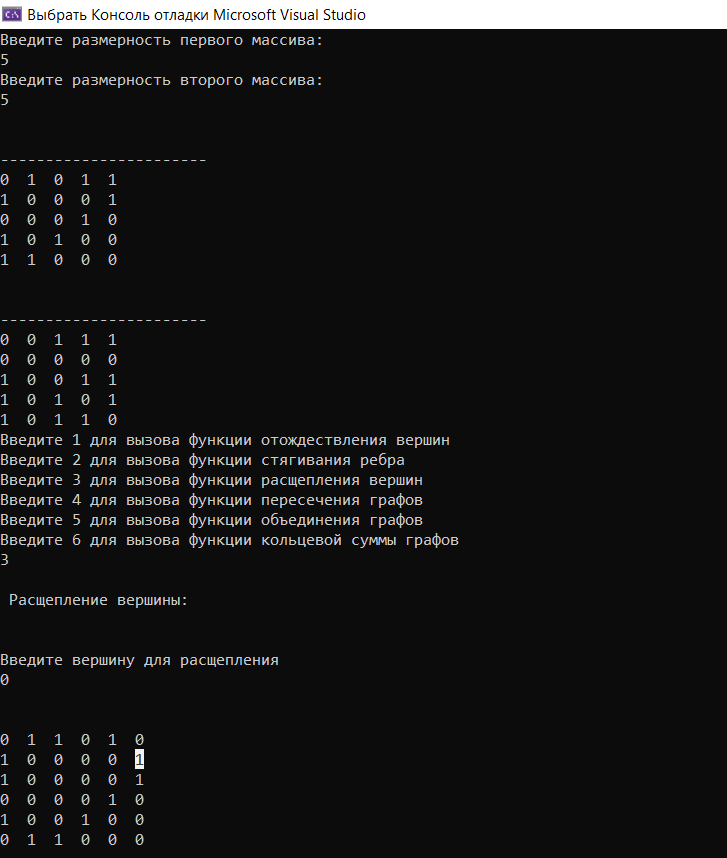


Рисунок 11 —Выполнение функции расщепления

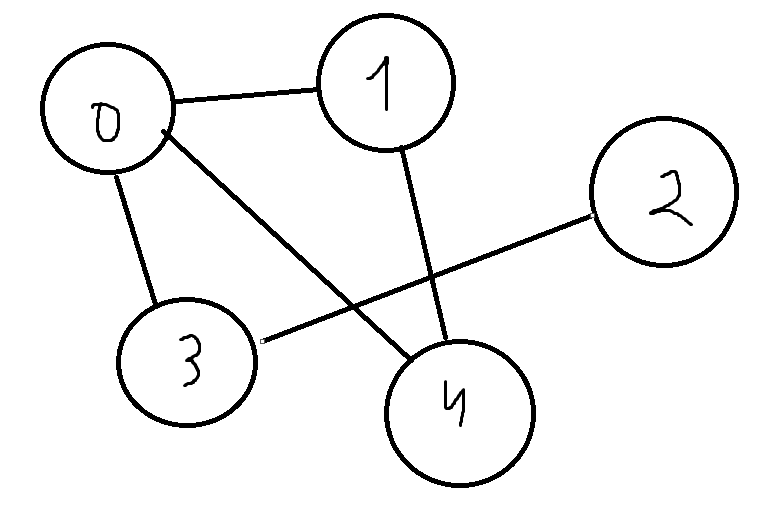


Рисунок 12 —Исходный граф

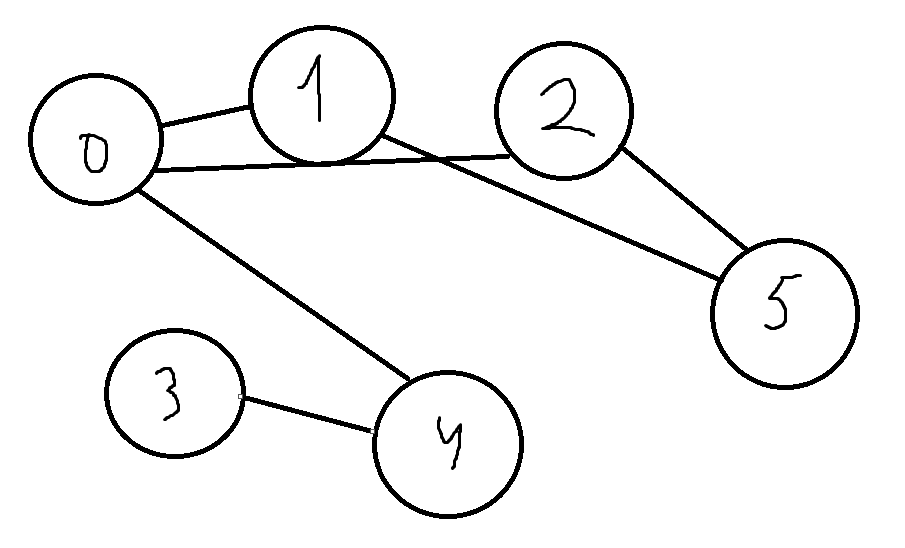


Рисунок 13 — Граф после преобразования

Вершина 0 расщепилась и в графе появилась новая вершина 1 (остальные вершины стали на 1 больше)

**Расчет функции пересечения:**

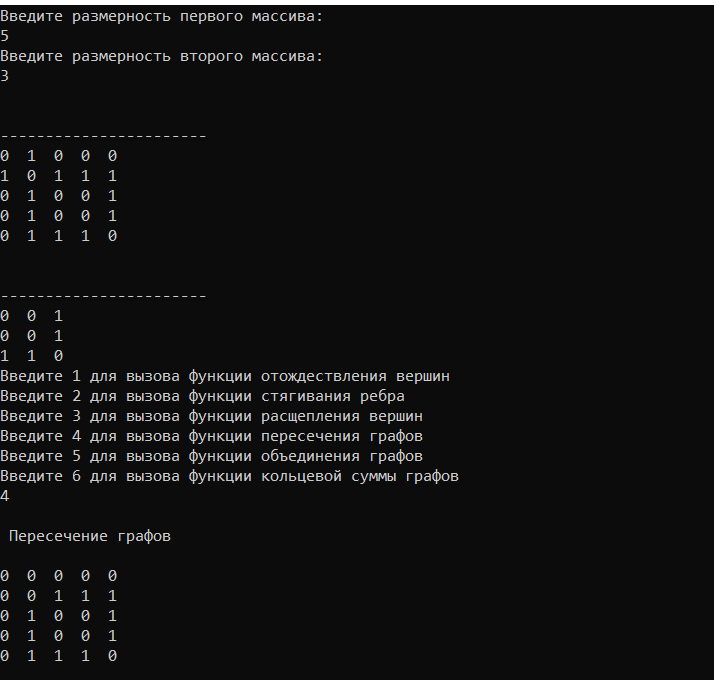


Рисунок 14 — Выполнение функции пересечения

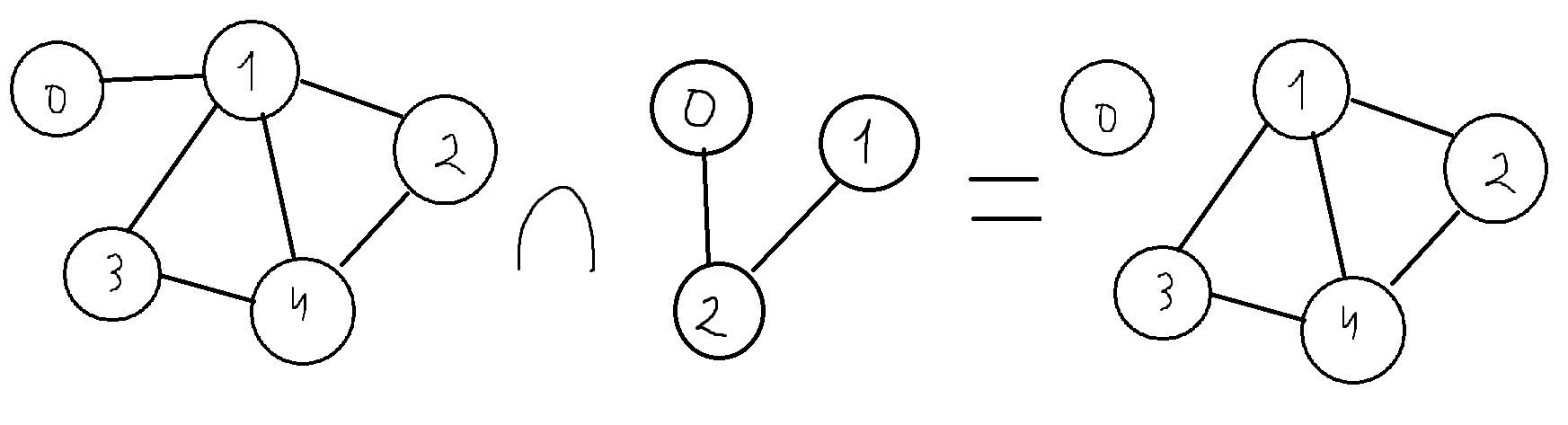


Рисунок 15 — Преобразование графов

**Расчет функции объединения:**

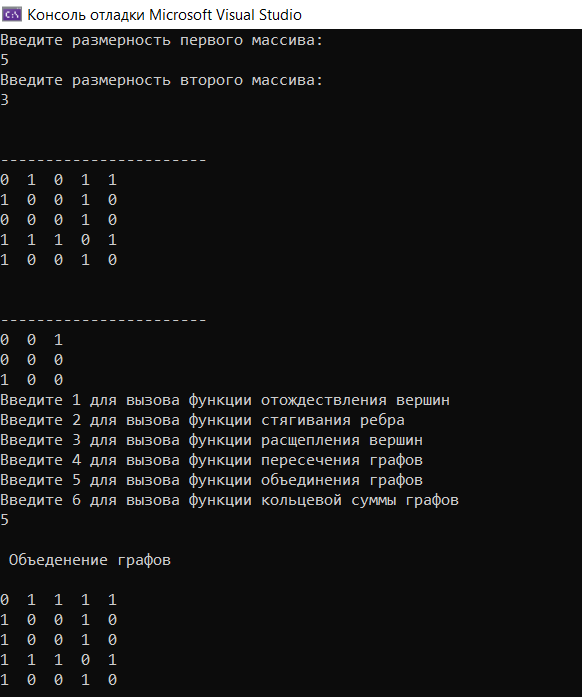


Рисунок 16 — Выполнение функции объединения

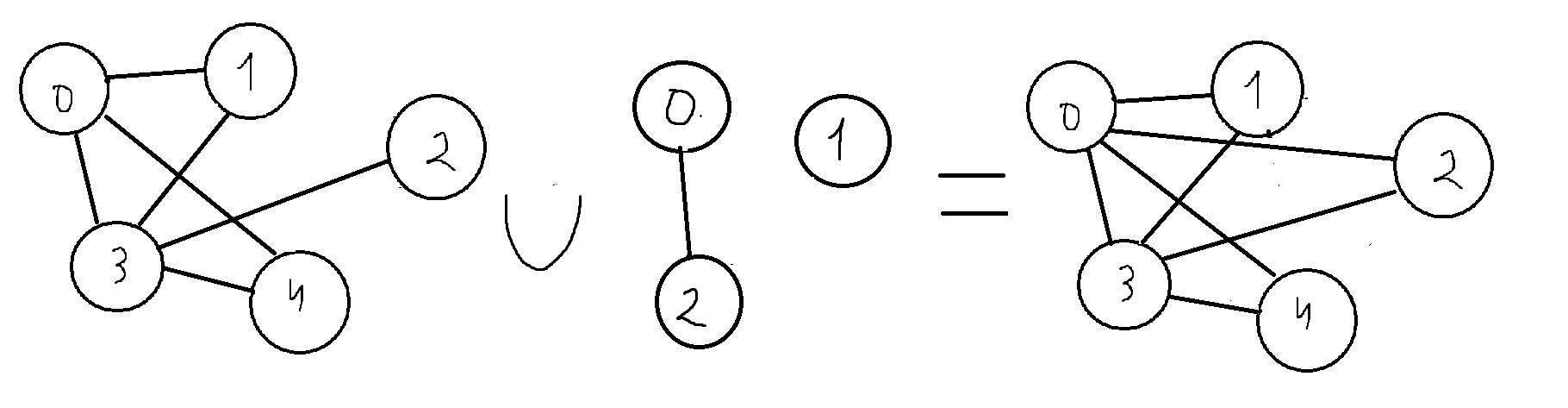


Рисунок 17 — Преобразование графов

**Расчет функции кольцевой суммы:**

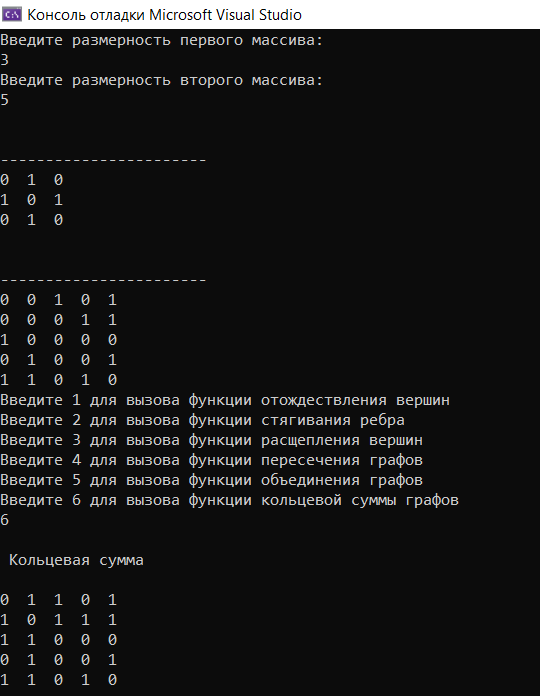


Рисунок 18 — Выполнение функции кольцевой суммы

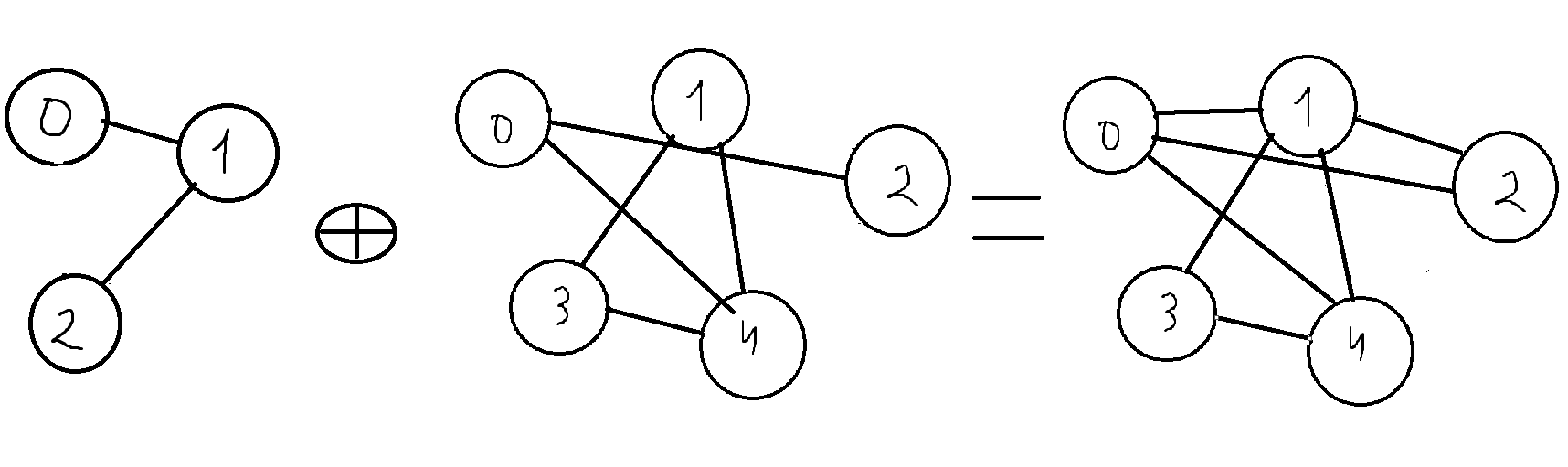


Рисунок 19 — Преобразование графов

**Расчет функции декартова произведения:**

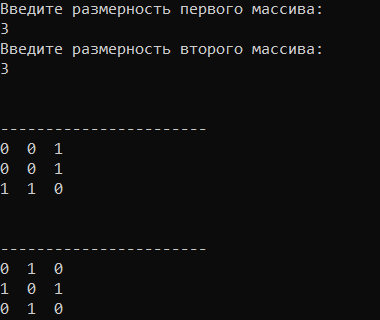
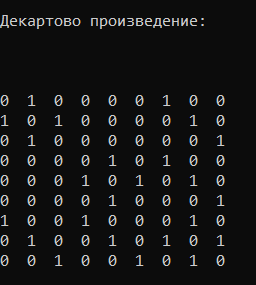


Рисунок 20 — Выполнение функции декартова произведения

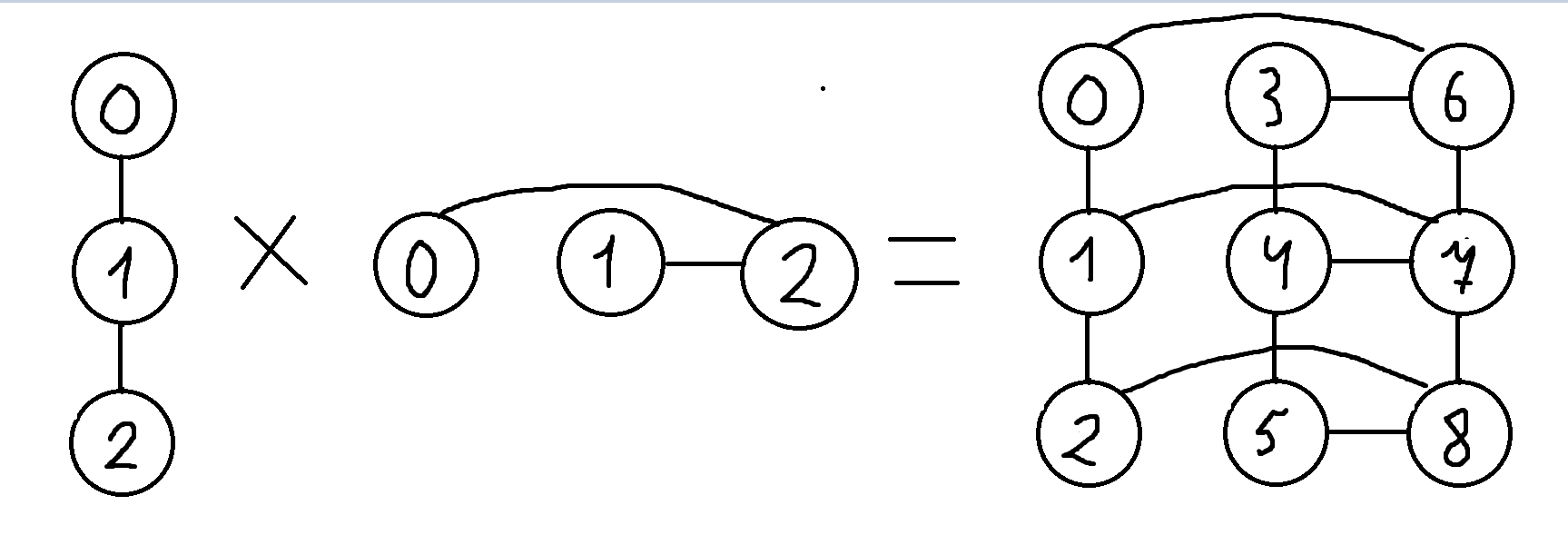


Рисунок 21 — Преобразование графов

***Заключение***

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритмы бинарный и унарных операций над графами в MicrosoftVisualStudio 2010. При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, основанных на теории графов.. Углублены знания языка программирования Cи. 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.

***Приложение А.***

***Листинг программы.***

#define \_CRT\_SECURE\_NO\_WARNINGS

#define \_CRT\_SECURE\_NO\_WARNING

#include "stdio.h"

#include <stdlib.h>

#include "conio.h"

#include "malloc.h"

#include <locale>

void decart(int\*\* mas1, int\*\* mas2, int men, int bol) {

printf("\n"); printf("\n");

printf("Декартово произведение:");

int\*\* result = (int\*\*)malloc((men \* bol) \* sizeof(int\*));

for (int i = 0; i < (men \* bol); i++) {

result[i] = (int\*)malloc((men \* bol) \* sizeof(int));

}

int ch1 = 0;

int ch2 = 0;

if (bol == men) {

for (int i = 0; i < (bol); i++) {

for (int j = 0; j < (bol); j++) {

if (mas1[i][j] == 1) { ch1 = ch1 + 1; }

if (mas2[i][j] == 1) { ch2 = ch2 + 1; }

}

}

}

if (ch1 < ch2) {

int\*\* mastemp = (int\*\*)malloc((bol) \* sizeof(int\*));

for (int i = 0; i < (bol); i++) {

mastemp[i] = (int\*)malloc((bol) \* sizeof(int));

}

for (int i = 0; i < (bol); i++) {

for (int j = 0; j < (bol); j++) {

mastemp[i][j] = mas1[i][j];

mas1[i][j] = mas2[i][j];

mas2[i][j] = mastemp[i][j];

}

}

}

for (int i = 0; i < (bol \* men); i++) {

for (int j = 0; j < (bol \* men); j++) {

result[i][j] = 0;

}

}

int mn = 0, i = 0, j = 0, z = 0, x = 0;

for (int k = 0; k < men; k++) {

for (z = 0, i = (i + mn); z < (bol); i++, z++) {

for (x = 0, j = (0 + mn); x < (bol); j++, x++) {

result[i][j] = mas1[z][x];

}

}

i = 0;

j = 0;

mn = mn + bol;

}

printf("\n"); printf("\n");

z = 0;

x = 0;

int c = 0;

int verI = 0;

int verJ = 0;

int mnoj = 0;

for ( i = 0; i < (men); i++, verI = verI + bol, verJ = verJ + bol) {

for (j = 0; j < (men); j++) {

if (mas2[i][j] == 1) {

mas2[j][i] = 0;

mnoj = (bol \* j) - (i \* bol);

for (c = 0; c < bol; c++) {

result[verI + c][verJ + mnoj + c] = 1;

}

}

}

mnoj = 0;

}

for (int i = 0; i < (bol \* men); i++) {

for (int j = 0; j < (bol \* men); j++) {

result[j][i] = result[i][j];

}

}

printf("\n"); printf("\n");

for (int i = 0; i < (bol \* men); i++) {

for (int j = 0; j < (bol \* men); j++) {

printf("%d ", result[i][j]);

}

printf("\n");

}

}

void Otoshdestvlenie(int\*\* mac, int o) {

printf("Отождествление вершин\n\n");

int v1, v2;

printf("Введите первую вершину для отождествления\n");

scanf("%d", &v1);

printf("Введите вторую вершину для отождествления\n");

scanf("%d", &v2);

for (int j = 0; j < o; j++) {

mac[v2][j] = (mac[v1][j] + mac[v2][j]);

if (mac[v2][j] == 2) {

mac[v2][j] = 1;

}

}

if (v2 <= (o / 2)) {

for (int i = 0; i < o; i++) {

for (int j = 0; j < o; j++) {

mac[j][i] = mac[i][j];

if (i == j) { mac[i][j] = 0; }

if ((i == j) && (i == v2)) { mac[i][j] = 1; }

}

}

}

else {

for (int i = o - 1; i >= 0; i--) {

for (int j = o - 1; j >= 0; j--) {

mac[j][i] = mac[i][j];

if (i == j) { mac[i][j] = 0; }

if ((i == j) && (i == v2)) { mac[i][j] = 1; }

}

}

}

int\*\* massa = (int\*\*)malloc((o) \* sizeof(int\*));

for (int i = 0; i < o; i++) {

massa[i] = (int\*)malloc((o) \* sizeof(int));

}

int\*\* massa2 = (int\*\*)malloc((o) \* sizeof(int\*));

for (int l = 0; l < o - 1; l++) {

massa2[l] = (int\*)malloc((o) \* sizeof(int));

}

for (int i = 0; i < o; i++) {

for (int j = 0; j < o; j++) {

if (i == v1) {

break;

}

if (j == v1) {

continue;

}

massa[i][j] = mac[i][j];

}

}

int k = 0, l = 0, i = 0, j = 0;

for ( i = 0; i < o; i++) {

for (j = 0; j < o; j++) {

if (k >= (o - 1)) {

l++;

k = 0;

}

if (massa[i][j] >= 0) {

massa2[l][k] = massa[i][j];

k++;

}

}

}

for (int i = 0; i < o - 1; i++) {

for (int j = 0; j < o - 1; j++) {

printf("%d ", massa2[i][j]);

}

printf("\n");

}

}

void Styagivanie(int\*\* mac1, int o1) {

printf("Стягивание ребра\n\n");

int v1, v2;

int Pr = 0;

while (Pr == 0) {

printf("Введите первую вершину для отождествления\n");

scanf("%d", &v1);

printf("Введите вторую вершину для отождествления\n");

scanf("%d", &v2);

if (((v2 != v1 + 1) || (v1 != v2 - 1)) && ((v1 != v2 + 1) || (v2 != v1 - 1))) {

printf("Стягивание ребра проводится только над смежными вершинами\n");

}

else {

Pr++;

}

}

for (int j = 0; j < o1; j++) {

mac1[v2][j] = (mac1[v1][j] + mac1[v2][j]);

if (mac1[v2][j] == 2) {

mac1[v2][j] = 1;

}

}

if (v2 <= (o1 / 2)) {

for (int i = 0; i < o1; i++) {

for (int j = 0; j < o1; j++) {

mac1[j][i] = mac1[i][j];

if (i == j) { mac1[i][j] = 0; }

}

}

}

else {

for (int i = o1 - 1; i >= 0; i--) {

for (int j = o1 - 1; j >= 0; j--) {

mac1[j][i] = mac1[i][j];

if (i == j) { mac1[i][j] = 0; }

}

}

}

int\*\* massa3 = (int\*\*)malloc((o1) \* sizeof(int\*));

for (int i = 0; i < o1; i++) {

massa3[i] = (int\*)malloc((o1) \* sizeof(int));

}

int\*\* massa4 = (int\*\*)malloc((o1) \* sizeof(int\*));

for (int i = 0; i < o1 - 1; i++) {

massa4[i] = (int\*)malloc((o1) \* sizeof(int));

}

for (int i = 0; i < o1; i++) {

for (int j = 0; j < o1; j++) {

if (i == v1) {

break;

}

if (j == v1) {

continue;

}

massa3[i][j] = mac1[i][j];

}

}

int k = 0, l = 0, i = 0, j = 0;

for (i = 0; i < o1; i++) {

for (j = 0; j < o1; j++) {

if (k >= (o1 - 1)) {

l++;

k = 0;

}

if (massa3[i][j] >= 0) {

massa4[l][k] = massa3[i][j];

k++;

}

}

}

printf("\n"); printf("\n"); printf("\n");

for (int i = 0; i < o1 - 1; i++) {

for (int j = 0; j < o1 - 1; j++) {

printf("%d ", massa4[i][j]);

}

printf("\n");

}

}

void Rascheplenie(int \*\*mac2, int o2) {

printf("Расщепление вершины:\n\n");

int Rv;

printf("\nВведите вершину для расщепления\n");

scanf("%d", &Rv);

int\* tempmas1 = (int\*)malloc((o2) \* sizeof(int));

for (int j = 0; j < o2; j++) {

if (j % 2 == 1) {

tempmas1[j] = mac2[Rv][j];

mac2[Rv][j] = 0;

mac2[j][Rv] = 0;

}

if (j % 2 == 0) {

tempmas1[j] = 0;

}

if (j == Rv) {

tempmas1[j] = 1;

}

}

int\*\* massa5 = (int\*\*)malloc((o2 + 1) \* sizeof(int\*));

for (int i = 0; i < (o2 + 1); i++) {

massa5[i] = (int\*)malloc((o2 + 1) \* sizeof(int));

}

int\*\* massaTrans = (int\*\*)malloc((o2 + 1) \* sizeof(int\*));

for (int i = 0; i < (o2 + 1); i++) {

massaTrans[i] = (int\*)malloc((o2 + 1) \* sizeof(int));

}

for (int i = 0; i < (o2 + 1); i++) {

for (int j = 0; j < (o2 + 1); j++) {

massa5[i][j] = mac2[i][j];

if (i == Rv) {

massa5[i][j] = tempmas1[j];

}

}

if (i == Rv) {

for (int k = o2; k > i; k--) {

mac2[k] = mac2[k - 1];

}

}

}

printf("\n");

for (int i = 0; i < o2 + 1; i++) {

for (int j = 0; j < o2 + 1; j++) {

massaTrans[i][j] = massa5[j][i];

}

}

for (int j = 0; j < (o2 + 1); j++) {

tempmas1[j] = massa5[Rv][j];

}

int\* tempmas2 = (int\*)malloc((o2) \* sizeof(int));

for (int i = 0; i < o2 + 1; i++) {

tempmas2[i] = massaTrans[Rv][i];

}

for (int i = 0; i < (o2 + 1); i++) {

for (int j = 0; j < (o2 + 1); j++) {

if (i == Rv) {

massaTrans[i][j] = tempmas1[j];

}

}

if (i == Rv) {

for (int k = o2; k > Rv; k--) {

massaTrans[k] = massaTrans[k - 1];

}

}

}

for (int i = 0; i < o2 + 1; i++) {

massaTrans[Rv][i] = tempmas2[i];

}

for (int i = 0; i < o2 + 1; i++) {

for (int j = 0; j < o2 + 1; j++) {

massa5[i][j] = massaTrans[j][i];

}

}

printf("\n");

for (int i = 0; i < o2 + 1; i++) {

for (int j = 0; j < o2 + 1; j++) {

if (i == j) {

massa5[i][j] = 0;

}

massa5[j][Rv] = massa5[Rv][j];

printf("%d ", massa5[i][j]);

}

printf("\n");

}

printf("\n\n\n");

}

void Peresechenie(int\*\* mac3, int\*\* mac4, int men, int bol) {

printf("Пересечение графов:\n\n");

int\*\* massa6 = (int\*\*)malloc((bol) \* sizeof(int\*));

for (int i = 0; i < bol; i++) {

massa6[i] = (int\*)malloc((bol) \* sizeof(int));

}

for (int i = 0; i < (bol); i++) {

for (int j = 0; j < (bol); j++) {

massa6[i][j] = mac4[i][j];

}

}

for (int i = 0; i < (men); i++) {

for (int j = 0; j < (men); j++) {

if ((mac3[i][j] == 1) && (mac4[i][j] == 1)) {

massa6[i][j] = 1;

}

else {

massa6[i][j] = 0;

}

}

}

for (int i = 0; i < (bol); i++) {

for (int j = 0; j < (bol); j++) {

printf("%d ", massa6[i][j]);

}

printf("\n");

}

printf("\n\n\n");

}

void obedenenie(int\*\* mac5, int\*\* mac6, int men, int bol) {

printf("Объеденение графов:\n\n");

int\*\* massa7 = (int\*\*)malloc((bol) \* sizeof(int\*));

for (int i = 0; i < bol; i++) {

massa7[i] = (int\*)malloc((bol) \* sizeof(int));

}

for (int i = 0; i < (bol); i++) {

for (int j = 0; j < (bol); j++) {

massa7[i][j] = mac6[i][j];

}

}

for (int i = 0; i < (men); i++) {

for (int j = 0; j < (men); j++) {

if ((mac5[i][j] == 1) || (mac6[i][j] == 1)) {

massa7[i][j] = 1;

}

else {

massa7[i][j] = 0;

}

}

}

for (int i = 0; i < (bol); i++) {

for (int j = 0; j < (bol); j++) {

printf("%d ", massa7[i][j]);

}

printf("\n");

}

printf("\n\n\n");

}

void Kolcevaya\_summa(int \*\*mac7, int \*\*mac8, int men, int bol) {

printf("Кольцевая сумма:\n\n");

int\*\* massa8 = (int\*\*)malloc((bol) \* sizeof(int\*));

for (int i = 0; i < bol; i++) {

massa8[i] = (int\*)malloc((bol) \* sizeof(int));

}

for (int i = 0; i < (bol); i++) {

for (int j = 0; j < (bol); j++) {

massa8[i][j] = mac8[i][j];

}

}

for (int i = 0; i < (men); i++) {

for (int j = 0; j < (men); j++) {

if ((mac7[i][j] == 1) || (mac8[i][j] == 1)) {

massa8[i][j] = 1;

}

if ((mac7[i][j] == 1) && (mac8[i][j] == 1)) {

massa8[i][j] = 0;

}

if ((mac7[i][j] == 0) && (mac8[i][j] == 0)) {

massa8[i][j] = 0;

}

}

}

for (int i = 0; i < (bol); i++) {

for (int j = 0; j < (bol); j++) {

printf("%d ", massa8[i][j]);

}

printf("\n");

}

}

void main() {

srand(time(NULL));

setlocale(LC\_ALL, "Russian");

int n;

int m;

printf("Введите размерность первого массива:\n");

scanf("%d", &n);

printf("Введите размерность второго массива:\n");

scanf("%d", &m);

int\*\* mas = (int\*\*)malloc(n \* sizeof(int\*));

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

mas[i] = (int\*)malloc(n \* sizeof(int));

}

int\*\* mas2 = (int\*\*)malloc((m + 1) \* sizeof(int\*));

for (int i = 0; i < (m + 1); i++) {

mas2[i] = (int\*)malloc((m + 1) \* sizeof(int));

}

int\*\* massave = (int\*\*)malloc(n \* sizeof(int\*));

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

massave[i] = (int\*)malloc(n \* sizeof(int));

}

int\*\* mas2save = (int\*\*)malloc((m + 1) \* sizeof(int\*));

for (int i = 0; i < (m + 1); i++) {

mas2save[i] = (int\*)malloc((m + 1) \* sizeof(int));

}

int\*\* mas2save1 = (int\*\*)malloc((m + 1) \* sizeof(int\*));

for (int i = 0; i < (m + 1); i++) {

mas2save1[i] = (int\*)malloc((m + 1) \* sizeof(int));

}

int\*\* mas2save2 = (int\*\*)malloc((m + 1) \* sizeof(int\*));

for (int i = 0; i < (m + 1); i++) {

mas2save2[i] = (int\*)malloc((m + 1) \* sizeof(int));

}

int\*\* massave1 = (int\*\*)malloc(n \* sizeof(int\*));

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

massave1[i] = (int\*)malloc(n \* sizeof(int));

}

int\*\* massave2 = (int\*\*)malloc(n \* sizeof(int\*));

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

massave2[i] = (int\*)malloc(n \* sizeof(int));

}

int\*\* massave3 = (int\*\*)malloc(n \* sizeof(int\*));

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

massave3[i] = (int\*)malloc(n \* sizeof(int));

}

int\*\* massave4 = (int\*\*)malloc(n \* sizeof(int\*));

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

massave4[i] = (int\*)malloc(n \* sizeof(int));

}

int\*\* massave5 = (int\*\*)malloc(n \* sizeof(int\*));

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

massave5[i] = (int\*)malloc(n \* sizeof(int));

}

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

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

mas[i][j] = rand() % 2;

if (i == j) { mas[i][j] = 0; }

}

}

printf("\n\n-----------------------\n");

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

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

mas[j][i] = mas[i][j];

if (i == j) { mas[i][j] = 0; }

printf("%d ", (mas[i][j]));

}

printf("\n");

}

for (int i = 0; i < m; i++) {

for (int j = 0; j < m; j++) {

mas2[i][j] = rand() % 2;

if (i == j) { mas2[i][j] = 0; }

}

}

printf("\n\n-----------------------\n");

for (int i = 0; i < m; i++) {

for (int j = 0; j < m; j++) {

mas2[j][i] = mas2[i][j];

if (i == j) { mas2[i][j] = 0; }

printf("%d ", (mas2[i][j]));

}

printf("\n");

}

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

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

massave[i][j] = mas[i][j];

massave1[i][j] = mas[i][j];

massave2[i][j] = mas[i][j];

massave3[i][j] = mas[i][j];

massave4[i][j] = mas[i][j];

massave5[i][j] = mas[i][j];

}

}

for (int i = 0; i < m; i++) {

for (int j = 0; j < m; j++) {

mas2save[i][j] = mas2[i][j];

mas2save1[i][j] = mas2[i][j];

mas2save2[i][j] = mas2[i][j];

}

}

int variant;

for (variant = 1; variant < 8; variant++) {

if (variant == 1) {

Otoshdestvlenie(massave, n);

}

if (variant == 2) {

Styagivanie(massave1, n);

}

if (variant == 3) {

Rascheplenie(massave2, n);

}

if (variant == 4) {

if (m <= n) {

Peresechenie(mas2save, massave3, m, n);

}

if (m > n) {

Peresechenie(massave3, mas2save, n, m);

}

}

if (variant == 5) {

if (m <= n) {

obedenenie(mas2save1, massave4, m, n);

}

if (m > n) {

obedenenie(massave4, mas2save1, n, m);

}

}

if (variant == 6) {

if (m <= n) {

Kolcevaya\_summa(mas2save2, massave5, m, n);

}

if (m > n) {

Kolcevaya\_summa(massave5, mas2save2, n, m);

}

}

if (variant == 7) {

if (n <= m) {

decart(mas2, mas, n, m);

}

if (n > m) {

decart(mas, mas2, m, n);

}

}

}

}