

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В. Г. Шухова»
(БГТУ им. В. Г. Шухова)



Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №4.3

по дисциплине: «Дискретная математика»

по теме: **Связность**

Выполнил/а: ст. группы ПВ-231

Чупахина София Александровна

Проверил: Рязанов Юрий Дмитриевич

Белгород, 2024

Содержание

Задание 1	3
Задание 2	4
Задание 3	8
Вывод	14

Задание 1

Текст задания: Реализовать алгоритм Краскала построения покрывающего леса.

Реализуем алгоритм в том виде, в котором это предложено сделать в лекции. В начале работы алгоритма создадим массив `bunches`, число элементов в котором равно числу элементов в графе вообще. Этот массив отображает, в каком букете какая вершина графа находится на данный момент формирования леса: если i -тый, начиная отсчет с нуля, элемент в массиве равен n , значит, вершина со значением $i+1$ находится в букете с номером n . Следует отметить, что в процессе формирования дерева номера букетов не обязательно идут от единицы до их конечного количества; их номера призваны только давать понять, в одном или разных букетах находятся вершины. Циклом пробегая все элементы массива, будем задавать им номера от 1 до максимального значения среди вершин графа. После этого с помощью двух вложенных циклов будем пробегать по элементам матрицы смежности данного графа, перебирая пары значений вершин. Для каждого элемента будем проверять, равен ли он единице (то есть смежна ли рассматриваемая пара вершин) и различны ли значения массива `bunches` на позициях, соответствующих этим элементам (то есть различны ли номера букетов, которым принадлежат рассматриваемые вершины). Если оба этих условия выполняются, то ребро между рассматриваемыми вершинами вносится в формируемый граф: в матрице смежности устанавливаются в 1 элементы, соответствующие данной паре вершин (без учета порядка). Затем происходит слияние букетов: запоминается номер букета для одной из вершин, а затем с помощью цикла все элементы массива `bunches` с этим значением заменяются на номер букета второй вершины.

```
1 #include "ДИСКРЕТКА../.. / ЛАБА
   3.1/bin_relations/bin_relation_definition_input_output.c"
2 #include "ДИСКРЕТКА../.. / ЛАБА
   3.1/bin_relations/bin_relations_operations.c"
3 #include "ДИСКРЕТКА../.. / ЛАБА 4.1/matrix/matrix.c"
4
5 typedef bin_relation graph;
6
7
8 graph graph_coveringByKruskal(graph A) {
9     graph KruskalForest = bin_relation_createEmpty(A.max_value);
10    int bunches[A.max_value];
11    for (int i = 0; i < A.max_value; i++) {
12        bunches[i] = i+1;
13    }
14    for (int i = 1; i <= A.max_value; i++) {
15        for (int j = 1; j <= A.max_value; j++) {
16            if (bin_relation_getValue(A, i, j) && bunches[i-1] !=
bunches[j-1]) {
17                bin_relation_changeValue(&KruskalForest, i, j, 1);
18                bin_relation_changeValue(&KruskalForest, j, i, 1);
19                int bunch_i = bunches[i-1];
20                for (int k = 0; k < A.max_value; k++) {
21                    if (bunches[k] == bunch_i) {
22                        bunches[k] = bunches[j-1];
```

```

23     }
24     }
25     }
26     }
27     }
28     return KraskalForest;
29 }

```

graphs/Kraskal.c

Задание 2

Текст задания: Используя алгоритм Краскала, разработать и реализовать алгоритм решения задачи (см. варианты заданий).

Задача для варианта 6: Найти все множества вершин, исключение которых из связного графа разбивает его на две связные компоненты, причем каждая компонента не содержит изолированных вершин.

Поступим следующим образом. Будем находить все возможные множества вершин для данного графа, для каждого из этих множеств строить граф, из которого эти вершины (условно) исключены, и для построенного графа определять, является ли он состоящим из двух связных компонент и нет ли в нем изолированных вершин. Первым делом, добавим в файл функции для работы с массивами — последовательностями целых чисел (в прошлых лабораторных работах мы представляли и упорядоченные последовательности, и множества вершин именно так). Если быть точнее, это функция вывода множества вершин на экран и функция, определяющая, входит ли в множество некоторый элемент.

```

1 #include "ДИСКРЕТКА../.. / ЛАБА
    3.1/bin_relations/bin_relation_definition_input_output.c"
2 #include "ДИСКРЕТКА../.. / ЛАБА
    3.1/bin_relations/bin_relations_operations.c"
3 #include "ДИСКРЕТКА../.. / ЛАБА 4.1/matrix/matrix.c"
4
5 #include "Kraskal.c"
6
7 void printSequence(int *W, int n) {
8     printf("(");
9     for (int j = 0; j < n; j++) {
10         printf("%d, ", W[j]);
11     }
12     printf("\b\b)\t");
13 }
14
15 bool isInSequence(int *W, int n, int value) {
16     for (int j = 0; j < n; j++) {
17         if (W[j] == value) {

```

```

18         return true;
19     }
20 }
21 return false;
22 }

```

graphs/task_6_final.c

Как мы будем перебирать все возможные множества вершин? Очевидно, это будут подмножества от множества всех вершин данного графа. Подмножество можно задать двоичным вектором, где количество разрядов равно количеству элементов исходного множества, и каждый разряд обозначает, входит ли в подмножество элемент исходного множества, соответствующий данному разряду (элементам присваиваются порядковые номера, и разряды двоичных векторов соотносятся с элементами по их номерам). Если множество имеет мощность $|M|$, то и двоичные вектора будут иметь длину $|M|$, а количество возможных значений этих векторов будет равно $2^{|M|}$. Значит, чтобы перебрать все возможные подмножества вершин, нам нужно перебрать все числа от 0 (все нули, ни один элемент не входит в подмножество) до $2^{|M|} - 1$ (все единицы, все элементы входят в подмножество). Впрочем, эти крайние значения можно и опустить, потому что понятно: не исключив ни одной вершины, мы не сможем разбить массив на две компоненты, а при исключении всех вершин полученный граф вообще компонент иметь не будет.

Для такой стратегии необходимо написать функцию, переводящую двоичный вектор (то есть обычное целое число) в последовательность целых чисел (значения отдельных разрядов в целом числе получаем с помощью побитовых операций):

```

1 ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
2 void getSequenceByVector(int vector, int *w, int *n) {
3     int cur_vert = 1;
4     (*n) = 0;
5     while (vector > 0) {
6         if (vector & 1) {
7             w[*n] = cur_vert;
8             (*n)++;
9         }
10        cur_vert++;
11        vector >>= 1;
12    }
13 }

```

graphs/task_6_final.c

Как уже было сказано, после получения каждого подмножества мы будем строить для него подграф исходного графа, в котором данные вершины удалены. В теле функции, выполняющей эту операцию, результирующий граф создается с тем же набором вершин (матрица смежности не меняет размерность), но без ребер. Все ребра исходного графа перебираются парой вложенных циклов; соответствующие им значения в матрице смежности переносятся в результирующий граф, только если ни одна из вершин не входит в последовательность, вершины из которой должны быть исключены.

```

1 ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
2 graph formGraphWithoutVertex (graph A, int *W, int n) {
3     graph B = bin_relation_createEmpty(A.max_value);
4     for (int i = 1; i <= A.max_value; i++) {
5         for (int j = 1; j <= A.max_value; j++) {
6             if (!isInSequence(W, n, i) && !isInSequence(W, n, j)) {
7                 bin_relation_changeValue(&B, i, j,
bin_relation_getValue(A, i, j));
8                 bin_relation_changeValue(&B, j, i,
bin_relation_getValue(A, j, i));
9             }
10        }
11    }
12    return B;
13 }

```

graphs/task_6_final.c

Для построенного графа необходимо определить два свойства. Первое — разбит ли он ровно на две связные компоненты? В этом нам и поможет алгоритм Краскала для построения покрывающего леса. Для одного исходного графа может быть построено несколько покрывающих деревьев/лесов, но количество ребер в них постоянно, не зависит от порядка, в котором перебирались ребра исходного графа при построении, и равно разности количества вершин и связных компонент исходного графа. Так что построив покрывающий лес по этому алгоритму, нужно лишь посчитать ребра в получившемся графе и сделать вывод, равно ли оно количеству вершин исходного графа минус два (ведь по условию задачи, граф должен быть разбит на две связных компоненты).

Алгоритм Краскала уже был реализован в задании 1. Остается только реализовать функцию подсчета ребер в графе.

```

1 ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
2 int count_edges(graph G) {
3     int result = 0;
4     for (int i = 1; i <= G.max_value; i++) {
5         for (int j = i+1; j <= G.max_value; j++) {
6             result += bin_relation_getValue(G, i, j);
7         }
8     }
9     return result;
10 }

```

graphs/task_6_final.c

Второе поставленное в задаче условие — отсутствие изолированных вершин. Здесь можно воспользоваться тем фактом, что строки в матрице смежности в данной реализации графов кодируются целыми числами (и разбиваются на нули и единицы побитовыми операциями). Если вершина изолирована, значит, она не имеет инцидентных ей ребер, соответствующая ей строка в матрице не содержит ни одной единицы и кодируется нулем. Для подсчета количества

изолированных вершин в графе нужно лишь перебрать в цикле все строки матрицы и вернуть количество нулевых.

Важно отметить: нам необходимо именно считать количество изолированных вершин, а не просто выносить вердикт о их наличии/отсутствии. Ведь после удаления вершин из графа (а фактически — удаления всех ребер, которым они были инциденты) информация о их существовании по-прежнему будет храниться в матрице смежности, и они будут считываться этим алгоритмом как изолированные. Значит, чтобы полученный граф удовлетворяло условию, необходимо, чтобы количество изолированных вершин в нем было равно количеству удаленных, но не превышало его.

```
1 ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
2 int amountOfIsolated(graph A) {
3     int result = 0;
4     for (int i = 1; i <= A.max_value; i++) {
5         if (A.values[i-1] == 0) {
6             result++;
7         }
8     }
9     return result;
10 }
```

graphs/task_6_final.c

Совместим все вышеописанные шаги и условия в результирующую функцию, перебирающую подмножества от исходного множества вершин графа, строящую подграф, из которого эти вершины удалены, и выводит на экран подмножество вершин, если полученный граф удовлетворяет условиям.

В целях уменьшения количества перебираемых подмножеств можно установить еще одно условие: начинать проверку условий для этого подмножества, только если количество вершин после удаления больше или равно четырем. Ведь меньшее количество вершин нельзя разбить на две связных компоненты так, чтобы хотя бы одна из них не оказалась изолированной.

```
1 ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
2 void vertexesToSplitGraphInTwo(graph A) {
3     int W[A.max_value];
4     int n;
5     for (int vector_of_set = 1; vector_of_set < (1<<A.max_value)-1;
6         vector_of_set++) {
7         getSequenceByVector(vector_of_set, W, &n);
8         if (A.max_value - n >= 4) {
9             graph copyA = formGraphWithoutVertex(A, W, n);
10            graph covering_copyA = graph_coveringByKruskal(copyA);
11
12            if (count_edges(covering_copyA) == A.max_value - 2 - n &&
13                amountOfIsolated(copyA) == n) {
14                printSequence(W, n);
15            }
16        }
17    }
18 }
```

```
15 }
16 }
```

graphs/task_6_final.c

Задание 3

Текст задания: Подобрать тестовые данные. Результат представить в виде диаграммы графа.

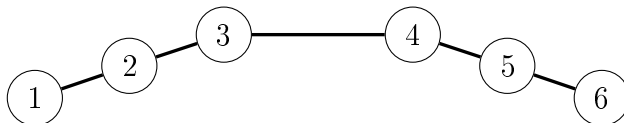
Вручную составим несколько графов, постаравшись отобразить в них как некоторые общие случаи, так и продемонстрировав показательные простые примеры. Затем будем создавать аналогичные графы в программе и запускать для них написанную функцию.

Для задания графа с помощью литералов определим в файле тестирования уже использовавшуюся ранее в лабораторных работах по графам функцию `graph_createFromMatrix`:






```
1 #include "ДИСКРЕТКА../.. / ЛАБА
    3.1/bin_relations/bin_relation_definition_input_output.c"
2 #include "ДИСКРЕТКА../.. / ЛАБА
    3.1/bin_relations/bin_relations_operations.c"
3 #include "ДИСКРЕТКА../.. / ЛАБА 4.1/matrix/matrix.c"
4
5 #include "task_6_final.c"
6
7 graph graph_createFromMatrix(matrix M) {
8     graph result = bin_relation_createEmpty(M.nRows);
9     for (int i = 1; i <= M.nRows; i++) {
10         for (int j = 1; j <= M.nRows; j++) {
11             bin_relation_changeValue(&result, i, j, M.values[i-1][j -
12             1]);
13         }
14     }
15     return result;
16 }
```

graphs/tests.c

Начнем с графа, вырождающегося в линейную последовательность.



Удаление его крайних вершин (имеющих степень связности 1) не приведет к разбиению на 2 компоненты; удаление вершин, смежных крайним, приведет к формированию компонент из изолированных вершин. Но вершины между теми, что смежны с крайними, удалять можно, как по одной, так и по несколько за раз, если они идут подряд; плюс, если удаление какой-то вершины приводит к формированию компоненты, где больше двух вершин, можно удалить в них крайние.

Удаленные вершины	Полученный граф
3	
4	
1, 4	
3, 4	
3, 6	

Функция выведет те же наборы вершин, что и в первом столбце таблицы:

```

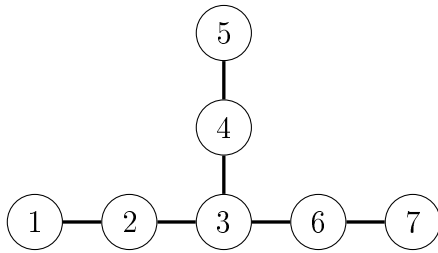
1 ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
2 int main() {
3     matrix M1 = createMatrixFromArray((int[]) {
4         0, 1, 0, 0, 0, 0,
5         1, 0, 1, 0, 0, 0,
6         0, 1, 0, 1, 0, 0,
7         0, 0, 1, 0, 1, 0,
8         0, 0, 0, 1, 0, 1,
9         0, 0, 0, 0, 1, 0
10    }, 6, 6);
11    graph G1 = graph_createFromMatrix(M1);
12    printf("Sets of vertexes, which deletion split graph G1 in two
13    components, without isolated vertexes:\n");
14    vertexesToSplitGraphInTwo(G1);
15    printf("\n");
16 }
```

graphs/tests.c

```

Sets of vertexes, which deletion split graph G1 in two components, without isolated vertexes:
(3)      (4)      (1, 4) (3, 4) (3, 6)
```

Следом рассмотрим граф, являющийся деревом.



Удаление его крайних вершин (имеющих степень связности 1) не приведет к разбиению на 2 компоненты; удаление вершин, смежных крайним, приведет к формированию компонент из изолированных вершин. Удаление корня дерева приведет к разделению графа на три компоненты — значит, необходимо при удалении корня дерева также удалять и все вершины, принадлежащие одной из полученных компонент.

Удаленные вершины	Полученный граф
1, 2, 3	
3, 4, 5	
3, 6, 7	

Функция выведет те же наборы вершин, что и в первом столбце таблицы:

```

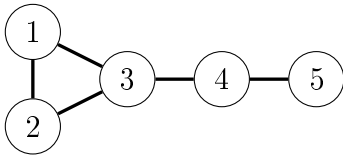
1 ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
2 int main() {
3     matrix M2 = createMatrixFromArray((int[]) {
4         0, 1, 0, 0, 0, 0, 0,
5         1, 0, 1, 0, 0, 0, 0,
6         0, 1, 0, 1, 0, 1, 0,
7         0, 0, 1, 0, 1, 0, 0,
8         0, 0, 0, 1, 0, 0, 0,
9         0, 0, 1, 0, 0, 0, 1,
10        0, 0, 0, 0, 0, 1, 0
11    }, 7, 7);
12    graph G2 = graph_createFromMatrix(M2);
13    printf("Sets of vertexes, which deletion split graph G2 in two
14    components, without isolated vertexes:\n");
15    vertexesToSplitGraphInTwo(G2);
16    printf("\n");

```

graphs/tests.c

```
Sets of vertexes, which deletion split graph G2 in two components, without isolated vertexes:
(1, 2, 3)      (3, 4, 5)      (3, 6, 7)
```

Следующий пример — граф, содержащий один цикл.



Для этого графа к нужному разделению приведет удаление только одной вершины — точки сочленения между циклом и остальной частью графа.

Удаленные вершины	Полученный граф
3	

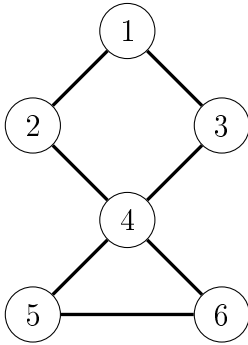
Функция выведет тот же набор вершин, что и в первом столбце таблицы:

```
1 ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
2 int main() {
3     matrix M3 = createMatrixFromArray((int[]) {
4         0, 1, 1, 0, 0,
5         1, 0, 1, 0, 0,
6         1, 1, 0, 1, 0,
7         0, 0, 1, 0, 1,
8         0, 0, 0, 1, 0,
9         0, 0, 0, 0, 1
10    }, 5, 5);
11    graph G3 = graph_createFromMatrix(M3);
12    printf("Sets of vertexes, which deletion split graph G3 in two
13    components, without isolated vertexes:\n");
14    vertexesToSplitGraphInTwo(G3);
15    printf("\n");
16 }
```

graphs/tests.c

```
Sets of vertexes, which deletion split graph G3 in two components, without isolated vertexes:
(3)
```

Рассмотрим также пример графа, содержащего 2 простых цикла; точка, соединяющая их — точка сочленения.



Удаление точки сочленения также приводит к разделению на две простых компоненты, но так как у одной из полученных компонент больше двух вершин (и она также вырождается в линейную последовательность), можно вместе с точкой сочленения удалить и крайние вершины полученной компоненты.

Удаленные вершины	Полученный граф
4	
2, 4	
3, 4	

Функция выведет те же наборы вершин, что и в первом столбце таблицы:

```

1 ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
2 int main() {
3     matrix M4 = createMatrixFromArray((int[]) {

```

```

4      0, 1, 1, 0, 0, 0,
5      1, 0, 0, 1, 0, 0,
6      1, 0, 0, 1, 0, 0,
7      0, 1, 1, 0, 1, 1,
8      0, 0, 0, 1, 0, 1,
9      0, 0, 0, 1, 1, 0
10     }, 6, 6);
11     graph G4 = graph_createFromMatrix(M4);
12     printf("Sets of vertexes, which deletion split graph G4 in two
13 components, without isolated vertexes:\n");
14     vertexesToSplitGraphInTwo(G4);
15     printf("\n");
16 }

```

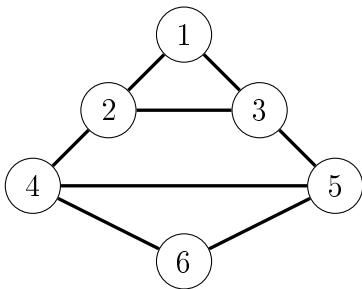
graphs/tests.c

```

Sets of vertexes, which deletion split graph G4 in two components, without isolated vertexes:
(4)      (2, 4) (3, 4)

```

Наконец, рассмотрим более сложный циклический граф.



В нем можно удалить только две пары вершин, чтобы полученное разделение удовлетворяло условию:

Удаленные вершины	Полученный граф
3, 4	
2, 5	

Функция выведет те же наборы вершин, что и в первом столбце таблицы:

```
1 ДИСКРЕТКАЛАБАДИСКРЕТКАЛАБАДИСКРЕТКАЛАБА
2 int main() {
3     matrix M5 = createMatrixFromArray((int[]) {
4         0, 1, 1, 0, 0, 0,
5         1, 0, 1, 1, 0, 0,
6         1, 1, 0, 0, 1, 0,
7         0, 1, 0, 0, 1, 1,
8         0, 0, 1, 1, 0, 1,
9         0, 0, 0, 1, 1, 0
10    }, 6, 6);
11    graph G5 = graph_createFromMatrix(M5);
12    printf("Sets of vertexes, which deletion split graph G5 in two
13    components, without isolated vertexes:\n");
14    vertexesToSplitGraphInTwo(G5);
15    printf("\n");
16 }
```

graphs/tests.c

```
Sets of vertexes, which deletion split graph G5 in two components, without isolated vertexes:
(3, 4) (2, 5)
```

Вывод

Не менее важное свойство, по которому можно классифицировать графы — связность; оно характеризует, существует ли как минимум один путь между любой парой вершин графа. Если такой путь существует для каждой пары вершин, то граф называется связным, иначе он разделен на несколько связных компонент: их вершины связаны между собой и не связаны с другими вершинами графа. Для каждого графа можно построить бинарное отношение связности, которое отображает, существует ли путь между каждой парой вершин. Такое отношение ищется путем получения транзитивного и рефлексивного замыкания отношения, представляющего матрицу смежности исходного графа. Отношение связности обладает свойством эквивалентности и множество вершин разбито по его классам эквивалентности так же, как оно разбито на связные компоненты.

Если между любой парой вершин связного графа существует только один путь (то есть в графе нет циклов), то такой граф называется деревом. Ациклический несвязный граф содержит несколько деревьев и называется лесом. Покрывающее (остовное) дерево некоторого графа — это дерево-подграф данного графа, содержащее все его вершины. Если изначальный граф несвязный, то используется понятие покрывающего (остовного) леса — множества покрывающих деревьев для каждой связной компоненты графа. По сути, процесс формирования такого дерева/леса сводится к удалению из исходного графа циклов (путем удаления одного из образующих цикл ребер). Для одного графа может существовать множество вариантов покрывающих деревьев/лесов, но каждое из них будет иметь одно и то же количество ребер: разность количества вершин и связных компонент исходного графа. Алгоритм Краскала описывает про-

цесс построения одного из таких деревьев/лесов, вводя понятие «букет» — множество ребер, принадлежащее одной и той же связной компоненте.

В ходе лабораторной работы реализовали на языке программирования алгоритм Краскала для построения покрывающего (остовного) леса; решили с его помощью более узкую задачу поиска всех подграфов, удовлетворяющих некоторому условию; подобрали тестовые данные для функции, решающей эту задачу, и убедились в корректности ее работы.