*УДК 519.16:621.39*

*Ткаченко В.Г., Клещёв А.О.*
*Ткаченко В.Г. Клещев А.О.*
*Tkachenco V.G.Kleshchev A.O.*

# МЕТОД ПРОВЕРКИ СВЯЗНОСТИ И ЦИКЛИЧНОСТИ ФРАГМЕНТА СЕТИ НА ОСНОВЕ МОНОТОННЫХ БУЛЕВЫХ ФУНКЦИЙ

# МЕТОД ПЕРЕВІРКИ ЗВ'ЯЗНОСТІ Й ЦИКЛІЧНОСТІ ФРАГМЕНТА МЕРЕЖІ НА ОСНОВІ МОНОТОННИХ БУЛЬОВИХ ФУНКЦІЙ

# THE METHOD FOR VERIFYING CONNECTIONS AND CYCLES IN FRAGMENT OF NETWORK USING MONOTONOUS BOOLEAN FUNCTIONS

***Аннотация.*** Разработан метод проверки связности и цикличности фрагмента сети. Этот метод использует одномерный массив для задания сети, элементы которого соответствуют ребрам сети. Для задания подграфа используется одномерный массив из номеров ребер массива сети. Эффективность достигается рассмотрением в массиве сети только тех элементов, номера которых входят в массив подграфа, а также использованием битовых операций И и ИЛИ над элементами массива сети и накопительной маской. Проведено сравнение этого метода с методами поиска в глубину и формирования матрицы достижимости по матрице смежности.

***Анотація***. Розроблено метод перевірки зв'язності й циклічності фрагмента мережі. Цей метод використовує одновимірний масив для задання мережі, елементи якого відповідають ребрам мережі. Для задання підграфа використовується одновимірний масив з номерів ребер масиву мережі. Ефективність досягається розглядом тільки тих елементів, номери яких включені до масиву підграфа, а також використанням бітових операцій И та ИЛИ над елементами масиву мережі та накопичувальною маскою. Проведено порівняння цього методу з методами пошуку в глибину та формування матриці досяжності за матрицею суміжності.

***Summary.*** The method for checking connectivity and cyclicity of the network fragment is developed. This method uses one-dimensional array, whose elements correspond to the edges of the network. The subgraph is defined by one-dimensional array filled with numbers of used network edges. Efficiency is achieved by analyzing only those edges, whose numbers are included in the subgraph array, and use of bitwise operators AND, OR on network elements and storage mask. Comparation of this method with the methods of depth-first search and the formation of the matrix reachability by the adjacency matrix is has been done.

В современных телекоммуникационных сетях состояние и структура могут многократно меняться в течении суток. В связи с этим возникает проблема в разработке эффективных методов для анализа и исследования фрагментов этих сетей. Для исследования и оптимизации телекоммуникационных сетей и систем очень часто применяется аппарат теории графов [1,2]. Одним из используемых понятий этой теории является дерево. Дерево – это связный подграф без циклов и петель. Частный случай дерева, которое включает все вершины графа, называется остовом (каркасом графа).

Широко известен метод проверки связности и цикличности на основе поиска в глубину [3], для проверки связности также применяют формирование матрицы достижимости по матрице смежности [4]. Реализация поиска в глубину предполагает прохождение всех еще не пройденных вершин вглубь, пока это возможно. Метод построения матрицы достижимости предполагает объединение строк матрицы смежности, которые относятся к вершинам смежным с вершинами уже найденными на предыдущем этапе.

Однако в методе поиска в глубину используется матрица инцидентности, за счет этого осложнены битовые операции и для хранения обязателен двухмерный массив. Также сам метод поиска в глубину чаще всего реализуется рекурсией, что подразумевает выделение большого количества памяти из стека и динамическое выделение памяти для хранения значений. В случае с методом формирования матрицы достижимости, для каждого нового подграфа необходимо формировать новую матрицу смежности, также затруднены проверки цикличности. Указанные недостатки снижают эффективность этих методов.

*Целью статьи* является разработка эффективного метода проверки связности и цикличности произвольной группы ребер телекоммуникационной сети.

Монотонная булева функция (МБФ) – это булева функция, значение которой не уменьшается при увеличении любого из её аргументов. В [5] показано, что МБФ можно представить в виде булевых выражений, где присутствуют только операции дизъюнкции и конъюнкции, но нет операции отрицания. Кроме того, переменные ни одной конъюнкции не входят полностью в другую. Такое представление называется минимальной дизъюнктивной формой. В [6] показано, какие МБФ используются в графах. Граф телекоммуникационной сети можно описать с помощью МБФ:

– инциденции (конъюнкции которые соответствую ребрам, а переменные конъюнкции – вершинам, соединяющим ребро);

– кликовой вершинной (соответствует полным подграфам из вершин) ;

– кликовой реберной (соответствует полным подграфам из ребер);

– вершинно-реберной (число конъюнкций равно числу вершин, переменные конъюнкции – ребра связанные с вершиной).

Разработанный метод определяет, является ли заданный подграф деревом (остовом), является ли он связным и содержит ли он циклы. Для представления графа используется транспонированная матрица инцидентности, что сильно упрощает поиск циклов и проверку связности. В методе используется только одна матрица без создания видоизмененных копий, за счет чего происходит экономия памяти. Поиск через матрицу инцидентности не использует рекурсию, за счет чего происходит экономия памяти и времени на рекурсивных переходах. За счет использования только простой арифметики и булевых функций, может быть реализован на любых микроконтроллерах и процессорах. Из недостатков следует отметить, что использование направленного графа невозможно.

Поскольку число ребер в дереве или остове графа $n - 1$, где $n$ – число вершин графа, входящих в предполагаемое дерево, либо число всех вершин графа при проверке остова, то все деревья и остовы данного графа являются подмножеством всех подграфов данного графа из $n - 1$ ребра.

Условием того, что выбранная связная группа ребер образует остов или дерево, будет наличие в ней всех вершин графа. Проверка осуществляется с помощью транспонированной матрицы инцидентности по выборке из всех ребер графа.

Разработанный метод состоит из следующих этапов:

1. Происходит пробег по всем ребрам в транспонированной матрице инцидентности, до тех пор, пока не будет найдено первое ребро, входящее в выбранный подграф согласно выборке из ребер. Первый найденный элемент сохраняется отдельно, в дальнейшем сохраненное значение будет являться маской использования вершин графа, для простоты - маска.

2. Согласно выборке из ребер выписываются все номера ребер в отдельный массив, кроме номера ребра, вершины которого были сохраненного в маску. Число ребер в массиве записывает отдельным числом – счетчиком массива.

3. Поочередно происходит операция И выбранных элементов матрицы по номерам из сохраненного массива с текущей маской. Если результат отличен от нуля, значит одна или более вершин ребра уже находится в маске и можно переходить к следующему шагу. Если результат равен значению элемента матрицы – исследуемый подграф содержит цикл и не является остовом (обе вершины ребра уже есть в маске), алгоритм завершает работу, так как подграф не является остовом или деревом. Если весь массив пройден, но не было значения отличного от 0, алгоритм завершает работу по причине отсутствия связности с некоторыми вершинами – подграф не является деревом или остовом.

4. К маске присваивается результат операции ИЛИ над маской и ребром из матрицы инцидентности (в маске появляется новая вершина), номер ребра в массиве номеров ребер замещается на последний элемент согласно счетчику элементов массива, счетчик элементов массива уменьшается на 1.

5. Если счетчик элементов массива указывает, что в массиве еще остались элементы, происходит возврат к этапу 3, начиная с первого элемента массива. Если счетчик достигнет значения 0 – алгоритм завершает работу, подграф является деревом, если число выбранных ребер соответствовало числу вершин минус один, то подграф является остовом.

Массив номеров ребер можно сформировать глобально, вне предложенной подпрограммы, или пересылать по ссылке, если предполагается многократный вызов функции для одного графа, так как число ребер остова всегда одинаковое и нет смысла многократно выделять и очищать память. В подпрограмму передается булевая функция используемых ребер либо массив номеров используемых ребер.

Опишем реализацию предложенного метода, выполненную на языках C# и Delphi. Транспонированная матрица инцидентности в нем реализуется одномерным массивом из целых чисел. Каждое ребро сети соответствует элементу массива. Два бита элемента соответствуют вершинам, которые соединяет ребро.

Приведем список глобально объявленных перемененных:

_usdList – глобально объявленный одномерный массив. Массив используется для хранения номеров входящих в подграф ребер. Массив не является динамическим, но считается, что длина массива указана в переменной vcnt, которая будет меняться в процессе работы. Нумерация массива от нуля. Этот массив позволяет избавится от необходимости постоянно переделывать транспонированную матрицу инцидентности либо необходимости проверки всех её компонент. Предусмотрено, что на входе может быть битовая последовательность val, или готовый массив номеров ребер. Размер массива больше либо равен числу ребер подграфа минус одно (еще одно сразу записывается в маску);

_connections – глобально объявленный одномерный массив (транспонированная матрица инцидентности), номер элемента массива соответствует ребру, значение элемента соответствует вершинам соединенных этим ребром (разряды соответствуют номеру вершины, 0 или 1 – соединена ли вершина ребром, в процессе работы матрица не меняется, поэтому может быть передана по ссылке или задана глобально). Нумерация от нуля. Приведенная реализация лучше работает с несортированным массивом.

Реализовано два метода задания подграфа: через массив номеров используемых ребер, либо через битовую маску используемых ребер. В первом случае алгоритм анализа работает непосредственно, во втором случае используется преобразование.

Ниже приведена реализация преобразования битовой маски в массив номеров ребер.

На входе функции число ребер в подграфе и битовая маска используемых ребер val (число ребер ограничено разрядностью переменной, разряды соответствуют ребрам в порядке их записи, а значения 1 или 0 – наличию или отсутствию определенного ребра в подграфе):

1. Выделяется память для маски вершин mask. Создается сдвиговая переменная sdv со стартовым значением 1 – она будет использована для пробега по битовой маске ребер. Создается счетчик для массива _usdList с названием vcnt и значением 0. Этот счетчик будет ограничивать область массива _usdList, с которой мы работает. Создается переменная i со значением, равным номеру первого элемента массива _connections.

2. До тех пор пока mask содержит значение 0, в цикле выполняются три операции:

– сравнение sdv и val операцией И, если значение отлично от 0, в mask записывается значение i-го элемента массива _connections;

– инкрементируется *i*;

– происходит побитовый сдвиг влево sdv.

3. Сравнение sdv и val операцией И. Если значение отлично от нуля, то в элемент массива _usdList с номером vcnt записывается значение *i*. Далее происходит инкремент vcnt.

4. Инкрементируется *i*, происходит побитовый сдвиг влево sdv.

5. Если vcnt меньше числа ребер подграфа минус 1, то происходит возврат к пункту 3.

6. Создание массива _usdList завершено. Вызываем подпрограмму анализа подграфа с необходимыми аргументами.

Далее приведена реализация алгоритма анализа.

На входе маска с первым элементом, массив номеров ребер и еще одно значение – режим работы mode, может иметь четыре состояния – “поиск дерева”, “поиск цикла”, “проверка связности”, “поиск циклов и проверка связности”.

Маску можно бы было определять уже в самом алгоритме, но её наличие позволяет произвольно указывать первый элемент и может быть использована для целей анализа подграфа или в целях ускорения работы (на работу влияют некоторые виды сортировки и задание в качестве маски ребра с наибольшим количеством связей).

Режим поиска дерева предполагает выход по обнаружению цикла либо достижению конца массива, за счет чего это самый быстрый вариант. Поиск цикла – проверка введенного подграфа на наличие циклов, алгоритм будет перебирать компоненты связности, пока не найдет цикл или не закончатся ребра. Проверка связности – все циклы игнорируются, выход происходит только по достижению конца массива (обнаружению разрыва) или обнаружению связанного графа. Режим

поиска циклов и проверки связности нужен для однозначного определения типа подграфа – он завершит работу только когда будет точно известно, является ли подграф связным и есть ли в нем циклы:

1. Создается переменная *i* со значением, равным номеру первого элемента массива _usdList. Булевым переменным unreach и hascicle присваиваем значение false – эти переменные будут отвечать за обозначение нахождения разрывов и циклов сети. В vcnt записываем число ребер подграфа минус 1.

3. В переменную elm записывается значение элемента _connections с номером из *i*-го элемента массива _usdList.

4. Если результат побитовой операции И над mask и elm вернул значение, равное 0, то инкрементируем *i* и переходим к пункту 13.

5. Если побитовое И над mask и elm возвращает значение elm, то в графе есть цикл.

Если нам нужно только определить, является ли подграф деревом или остовом - прерываем выполнение подпрограммы с выводом “найден цикл”, подграф не является деревом. Если же нам согласно mode необходимо проверить связность всех вершин подграфа - продолжаем выполнение, hascicle присваиваем true (значение по умолчанию false).

6. Присваиваем mask значение побитовой операции mask ИЛИ elm.

7. Декрементируем vcnt, записываем в *i*-й элемент _usdList значение элемента с номером vcnt.

8. Возвращаем *i* на начало массива.

9. Если элемент usdList с номером *i* является последним элементом массива согласно vcnt, и в массиве остались не использованные элементы (vcnt > 0), то подграф не является деревом так как не все вершины соединены. Если нам необходима только проверка, является ли подграф деревом – прерываем выполнение со значением “нет связности”, подграф не является деревом. Если согласно mode нужно определить, есть ли в подграфе циклы, присваиваем к маске значение элемента _connections с номером из первого элемента массива _usdList. Сбрасываем значение *i* на начало массива в первый элемент _usdList, переписываем значение последнего, уменьшаем значение счетчика vcnt. К unreach присваиваем значение true.

10. Если vcnt больше 0, то возвращаемся к п. 7.

11. Если unreach и hascicle не установлены, все элементы использованы, то подграф – дерево или остов. Иначе возвращаем значения в соответствии с выбранным режимом.

В целях оптимизации можно добавить дополнительное условие выхода – как только unreach и hascicle установлены в true – завершать выполнение. В процессе отладки можно использовать значение mask для повторной проверки правильности решения, для остова оно будет содержать значение $2^{reb+1}$-1 – это значит, что все вершины задействованы (предполагается, что ребер больше чем вершин). Так же его можно использовать для определения, является ли подграф остовом или только деревом, – если подграф не остов, то значение будет отличным. С небольшой модификацией, алгоритм может считать суммарную стоимость дерева, если его ребра имеют вес, – во время присвоения значений к маске суммировать вес того же ребра в отдельную переменную.

За счет применения битовых функций в транспонированной матрице инцидентности представление сети занимает намного меньше места и программный алгоритм сильно упрощен, но эти операции вводят ограничение на максимальный размер сети – число ребер не должно превышать разрядность используемых переменных, что для большинства аппаратных систем составляет 32, для персональных компьютеров 64 и 128. Для обхода ограничения можно применять длинную арифметику.

Приведем примеры работы предложенного метода в различных режимах для графа телекоммуникационной сети, изображенного на рис. 1.

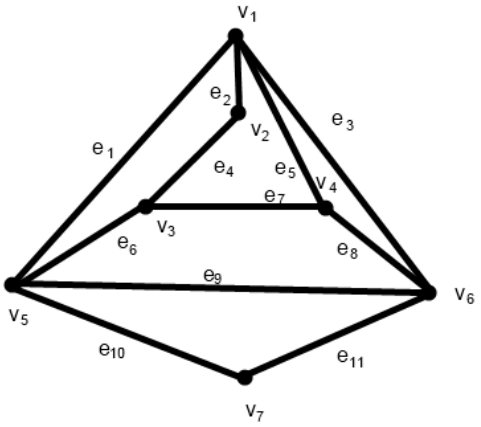


Рисунок 1 – Пример графа

Граф в виде МБФ инциденции имеет вид:

$$v_1v_5 \vee v_1v_2 \vee v_1v_6 \vee v_2v_3 \vee v_1v_4 \vee v_3v_5 \vee v_3v_4 \vee v_4v_6 \vee v_5v_6 \vee v_5v_7 \vee v_6v_7.$$

Транспонированная матрица инцидентности графа представлена ниже.

Номер строки соответствует номеру ребра, с

нумерацией с единицы, нумерация вершин соответствует нумерации столбцов справа налево начиная с единицы. Каждую строку матрицы также можно представить в виде МБФ. Например, третья строка может быть представлена как $v_1 \vee v_6$. Аналогично представляется маска mask.

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Ниже приведен пример работы алгоритма в различных режимах на приведенном графе.

Анализ подграфа $e_1 \vee e_2 \vee e_3 \vee e_4 \vee e_5 \vee e_{11}$ (нумерация ребер с единицы). Для данного подграфа все режимы работают одинаково, так он представляет собой дерево.

Согласно значению val 10000011111 в маску записывается 0010001 (вершины 1 и 5), в массив используемых ребер записываются значения 2, 3, 4, 5 и 11.

Таблица 1 – Пошаговое выполнение цикла сравнения маски с элементом матрицы инцидентности операцией И

| № п/п | mask | № ребра | Значение ребра | Результат И | Массив номеров ребер |
|---|---|---|---|---|---|
| 1 | 0010001 | 2 | 0000011 | 0000001 | 11, 3, 4, 5 |
| 2 | 0010011 | 11 | 1100000 | 0000000 | 11, 3, 4, 5 |
| 3 | 0010011 | 3 | 0100001 | 0000001 | 11, 5, 4 |
| 4 | 0110011 | 11 | 1100000 | 0100000 | 4, 5 |
| 5 | 1110011 | 4 | 0000110 | 0000010 | 5 |
| 6 | 1110111 | 5 | 0001001 | 0000001 | - |

Результат этапов 1, 3 … 6 отличался от нуля, поэтому на этом этапе менялось значение маски и удалялось заменой содержимое массива ребер. Так как в массиве номеров ребер не осталось значений, то программа выдаст ответ “дерево” независимо от режима. Так как число задействованных вершин равно числу вершин всего графа, то подграф является остовом графа.

В следующих двух таблицах анализируется подграф $e_1 \vee e_3 \vee e_4 \vee e_7 \vee e_9 \vee e_{10}$, этот подграф состоит из двух компонент связности, одна из которых имеет цикл.

Таблица 2 – Режим поиска “Поиск дерева”

| № п/п | mask | № ребра | Значение ребра | Результат И | Массив номеров ребер |
|---|---|---|---|---|---|
| 1 | 0010001 | 3 | 0100001 | 0000001 | 10, 4, 7, 9 |
| 2 | 0110001 | 10 | 1010000 | 0010000 | 9, 4, 7 |
| 3 | 0110011 | 9 | 0110000 | 0110000 | 9, 4, 7 |

Согласно значению val 01101001101 в маску записывается 0010001 (вершины 1 и 5), в массив используемых ребер записываются значения 3, 4, 7, 9 и 10.

На этапе 3 работа была завершена из-за обнаруженного цикла (значение ребра соответствует результату операции И). В этом режиме алгоритм прекращает выполнение по достижению конца массива (нет связности) или нахождению цикла.

Таблица 3 – Режим поиска “Проверка связности”

| № п/п | mask | № ребра | Значение ребра | Результат И | Массив номеров ребер |
|---|---|---|---|---|---|
| 1 | 0010001 | 3 | 0100001 | 0000001 | 10, 4, 7, 9 |
| 2 | 0110001 | 10 | 1010000 | 0010000 | 9, 4, 7 |
| 3 | 1110001 | 9 | 0110000 | 0110000 | 7, 4 |
| 4 | 1110001 | 7 | 0001100 | 0000000 | 7, 4 |
| 5 | 1110011 | 4 | 0000110 | 0000000 | 7, 4 |

В режиме проверки связности циклы игнорируются, поэтому на этапе 3 ребро составившее цикл удалено из списка, но работа продолжена. На этапе 5 был найден разрыв и завершена работа с уведомлением о наличии разрыва. В предложенной реализации режим будет производить выход по обнаружению разрыва, однако его легко модифицировать для подсчета всех компонент связности.

Анализ подграфа $e_1 \vee e_3 \vee e_4 \vee e_7 \vee e_{10}$ в режиме поиска циклов.

Для этого случая val равен 01001001101, маска 0010001, в массив записываются 3,4,7 и 10.

Таблица 4 – Режим поиска “поиск циклов”

| № п/п | mask | № ребра | Значение ребра | Результат И | Массив номеров ребер |
|---|---|---|---|---|---|
| 1 | 0010001 | 3 | 0100001 | 0000001 | 10, 4, 7 |
| 2 | 0110001 | 10 | 1010000 | 0010000 | 7, 4 |
| 3 | 1110001 | 7 | 0001100 | 0000000 | 7, 4 |
| 4 | 1110001 | 4 | 0000110 | 0000000 | 4 |
| 5 | 0001100 | 4 | 0000110 | 0000100 | - |

На этапе 3 был достигнут конец массива, но в этом режиме алгоритм продолжает работу, выход произойдет только по обнаружению цикла или использованию всех элементов массива ребер. В процессе поиска не было найдено циклов, однако был найден разрыв и алгоритм выдаст сообщение о наличии разрыва. Данный режим легко модифицировать для поиска всех независимых циклов в подграфе.

Режим “поиск циклов и проверка связности” совмещает два предыдущих примера.

Представленная реализация может определить только общие свойства связности или цикличности подграфа, но в сетях может возникнуть необходимость найти все компоненты связности и число независимых циклов в анализируемом фрагменте сети. Нахождение этих элементов также возможно с помощью данного алгоритма после небольшого изменения.

Необходимы счетчики элементов связности и счетчики найденных циклов. Для этого вводится двухмерный массив для хранения числа ребер и независимых циклов в компоненте связности – номер столбца соответствует номеру компоненты связности, значение первой строки соответствует числу ребер в компоненте, а вторая строка – числу обнаруженных в нём циклов. Также нужен двухмерный массив из двух столбцов, столбцы соответствуют номерам ребер, значение первой строки – номер компоненты связности, которой принадлежит ребро, вторая показывает, образовало ли ребро цикл (при такой организации подграф можно быстро переделать в дерево, удалив лишние ребра).

По нахождению цикла необходимо не завершать работу, а инкрементировать счетчик циклов. По достижению конца массива нужно записывать значение счетчика циклов в элемент, соответствующий компоненте связности, инкрементировать счетчик связности и сбрасывать счетчик циклов. При этом массив для компонент связности можно создавать с фиксированным размером, так как число компонент связности не превышает число вершин графа. По мере использования ребер нужно помечать в отдельном массиве, какому номеру компоненты связности соответствует ребро и образовало ли оно цикл.

Таблица 5 – Анализ подграфа $e_1 \vee e_3 \vee e_4 \vee e_7 \vee e_9 \vee e_{10}$.

| № п/п | mask | № ребра | Значение ребра | Результат И | Компонента связности/цикл | Массив номеров ребер |
|---|---|---|---|---|---|---|
| 1 | 0010001 | 3 | 0100001 | 0000001 | 1/0 | 10, 4, 7, 9 |
| 2 | 0110001 | 10 | 1010000 | 0010000 | 1/0 | 9, 4, 7 |
| 3 | 1110001 | 9 | 0110000 | 0110000 | 1/1 | 7, 4 |
| 4 | 1110001 | 7 | 0001100 | 0000000 | - | 7, 4 |
| 5 | 1110011 | 4 | 0000110 | 0000000 | - | 7, 4 |
| 6 | - | 7 | 0001100 | - | 2/0 | 4 |
| 7 | 0001100 | 4 | 0000110 | 0000100 | 2/0 | - |

Вначале считаем, что весь подграф связный и является компонентой связности с номером 1. Также считаем, что подграф не имеет циклов, а сохраненное в маску ребро считаем частью первой компоненты связности. На третьем этапе был обнаружен цикл, инкрементируется счетчик связности текущей (первой) компоненты, ребро 3 помечается как образовавшее цикл. На 5 этапе достигнут конец массива, то есть выделена первая компонента связности. Затем переписываем значение ребра 7 в маску и считаем, что оно принадлежит второй компоненте связности, в которой еще нет циклов.

На выходе получили, число компонент связности, равное 2 и два массива. Первый массив компонент связности, указывающий на число ребер и циклов в каждой компоненте связности (первая компонента – 4 ребра и цикл, вторая – два ребра):

$$\begin{pmatrix} 4 & 2 \\ 1 & 0 \end{pmatrix}$$

Второй массив ребер, показывающий, какой компоненте принадлежит ребро и образует ли оно цикл (номер столбца соответствует номеру ребра, первая строка – номер компоненты связности, все не задействованные в подграфе ребра обозначены 0-й компонентой связности, вторая строка – образовало ли ребро цикл):

$$\begin{pmatrix} 1 & 0 & 1 & 2 & 0 & 0 & 2 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Разработанная программа на основе МБФ в процессе работы всегда занимает одинаковую область памяти (фиксированный объём памяти выделяется на старте), занимаемая память растет линейно с ростом размера подграфа. В рекурсивном методе память выделяется динамически, используемый объем быстро растет. Объем памяти, требуемый для рекурсивного алгоритма [3], в любом случае значительно превосходил объем алгоритма на основе МБФ.

Сравнение работы в режиме “Поиск дерева” с алгоритмом на основе рекурсивного поиска в глубину [3] показало, что по скорости работы с графами до 25 … 30 ребер, эти алгоритмы отличаются незначительно. Сравнение работы алгоритма в режиме “Проверка связности” (поиск достижимых вершин) с поиском в глубину, показало, что скорость предложенного алгоритма существенно выше в разветвленных сетях. Работа предложенного алгоритма на основе МБФ в этой ситуации аналогична поиску в ширину (перебирает все ребра смежные с уже найденными) и намного быстрее находит пути, делает вывод о связанности подграфа. Однако поиск в глубину показывает намного большую эффективность при разборе длинных, непрерывных маршрутов. В графе из 20 ребер и 10 вершин было проверено 167960 различных подграфов по 9 ребер, при анализе одного подграфа в среднем было выполнено 49,4 относительных операций для алгоритма на основе МБФ и 171,7 операций для поиска в глубину. Относительная производительность предложенного алгоритма в этом примере в 3,48 раза выше. Для графа из 7 ребер и 5 вершин это отношение составляло 2,83 раза, из чего можно сделать вывод, что количество операций поиска в глубину растет быстрее с увеличением размера графа.

Алгоритм, приведенный в [4] удобен для анализа графа в целом, пригоден для учета направленных ребер, но не удобен для анализа большого количества подграфов из-за необходимости постоянно переформировывать исходную матрицу смежности. Также сложно организовать битовые операции из-за того, что в матрице смежности встречаются три различных значения.

В заключение отметим следующее. Результатом работы является метод анализа цикличности и связности сети либо её фрагментов с быстрым выходом по обнаружению неисправности. Метод предполагается использовать для быстрой проверки связности сети и наличия циклов, к примеру на этапе настройки или проектирования как виртуальных, так и реальных сетей (проверка всех маршрутов по умолчанию, анализ достижимости объектов…), где могут возникнуть частые изменения логической структуры. Анализ работы и сравнение с популярным методом поиска в глубину показали, что разработанный метод наиболее эффективен в разветвленных сетях. Вторым результатом является разработка модификации данного метода. В этой модификации не происходит выхода при встрече несвязности или цикличности в фрагменте сети, но зато собирается дополнительная информация об этом фрагменте. В частности находятся все компоненты связности и входящие в них ребра. Кроме того, для каждой компоненты связности находится остов и ребра, образующие с этим остовом независимые циклы.

***Литература***


1. *Свами М.* Графы, сети и алгоритмы / М. Свами, К. Тхуласираман. – М.: Мир, 1984. – 455 с.
2. *Захарченко Н.В.* Оптимизация и моделирование систем связи: учеб. пособ. / Н.В. Захарченко, Н.А. Князева; – Одесса: Электротехн. ин-т связи. –1990. – Ч. 2. – 76 с.
3. *Костюкова Н.И.* Графы и их применение. Комбинаторные алгоритмы для программистов /Костюкова Н.И. – М.:Бином, Интернет-Университет Информ. технологий, 2007. – 311 с.
4. *Кристофидес Н.* Теория графов. Алгоритмический подход / Кристофидес Н**.** – М.:Мир, 1978. – 432 с.
5. *Ткаченко В.Г*. Отказы цифровых схем и представления монотонных булевых функций / В.Г. Ткаченко //Наукові праці ОНАЗ ім. О.С. Попова. – 2006. – № 2. – С. 45 – 69.
6. *Ткаченко В.Г.* Методы прямого и обратного перебора остовов графов телекоммуникационных сетей/ В.Г. Ткаченко, Д.Г. Ларин. //Труды одесского политехнического университета. – 2011. – № 2 (36).