Практическое занятие № 4.

ОЦЕНКА СТРУКТУРНОЙ СЛОЖНОСТИ ПРОГРАММ

Время выполнения практического занятия (аудиторные часы) – 4 часа.

Время самостоятельной работы студента (дополнительные часы) – 4 часа.

Цель работы: изучить порядок оценки структурной сложности программного обеспечения.

Ключевые понятия, которые необходимо знать: надежность, метрики, критерии оценки, структурная сложность, граф, блок-схема.

Оборудование и программное обеспечение: работа выполняется на ПЭВМ типа IBM PC.

1. Теоретические сведения

1.1. Критерии структурной сложности программ.

Понятие структурной сложности программ.

Структурная сложность программ определяется:

- количеством взаимодействующих компонентов;
- числом связей между компонентами;
- сложностью взаимодействия компонентов.

При работе программы многообразие ее поведения и разнообразие связей между ее входными и результирующими данными в значительной степени определяется *набором маршрутов,* по которым исполняется программа. При этом под маршрутом следует понимать чередование последовательностей вершин и дуг графа управления. Уже давно установлено, что *сложность программных модулей* связана не столько с размером программы, определяющей количество выполняемых команд, сколько с *числом маршрутов* ее исполнения и их *сложностью.*

При создании качественной программы основную сложность ее разработки определяют маршруты возможной обработки данных, которые должны быть тщательно проверены. Такую метрику сложности, связанную с анализом маршрутов, можно использовать для оценки трудоемкости тестирования и сопровождения модуля, а также для оценки **потенциальной надежности** его функционирования. Проведенные исследования подтверждают достаточно высокую адекватность использования структурной сложности программ для оценки трудоемкости тестирования, вероятности ненайденных ошибок и затрат на разработку программных модулей в целом.

Совокупность маршрутов исполнения программного модуля условно можно разделить **на две группы**:

- вычислительные маршруты;

- маршруты принятия логических решений.

Группа *вычислительных маршрутов* объединяет в себе маршруты арифметической обработки данных и предназначена для непосредственного преобразования величин, являющихся элементарными результатами измерения каких-либо характеристик (переменных).

Для проверки вычислительных маршрутов можно сформировать достаточно простую стратегию их проверки. Во всем диапазоне преобразования входных переменных нужно выбрать несколько характерных точек, в которых проверяется работоспособность и корректность работы программы. К таким контрольным точкам можно отнести предельные значения, значения в точках разрыва, несколько промежуточных значений.

Сложность вычислительных маршрутов можно оценить следующей формулой:

$$S_1 = \sum_{i=1}^{m} \sum_{j=1}^{l_i} v_j \qquad (1)$$

где: m - количество маршрутов исполнения программы; l_i - число данных, обрабатываемых в i-м маршруте; v_j - число значений обрабатываемых данных j типа ($2 < v_j < 5$).

Расчет сложности программы по этой формуле имеет высокую неопределенность из-за отсутствия конкретных требований в выборе количества значений переменной v_j при изменении входных данных. Поэтому применение этой формулы для оценки сложности весьма затруднительно.

В связи с тем, что удельный вес (доля) вычислительной части во многих сложных программных комплексах обработки информации относительно невелика (общее число арифметических операций не выходит за пределы 5-10 %), вычислительные маршруты не играют доминирующей роли в определении структурной сложности программ. Это связано с тем, что проведение вычислений, даже весьма разветвленных, все же подчиняется определенным правилам, несущественно влияющим на логику проведения этих вычислений.

Группа *маршрутов принятия логических решений* объединяет пути, отражающие логику выполнения программы, которая может изменять последовательность выполнения команд, переводить управление на удаленные участки программного модуля или даже досрочно завершать его выполнение. Все эти факторы могут весьма значительно влиять на сложность программы.

Сложность маршрутов принятия логических решений оценивается формулой:

$$S_2 = \sum_{i=1}^{m} p_i \qquad (2)$$

где p_i - число ветвлений или число проверяемых условий в i-м маршруте.

Поскольку большинство программных продуктов сочетает в себе как вычислительные маршруты, так и маршруты принятия логических решений, устанавливают способ определения общей сложности программы.

Общую сложность программы можно рассчитать по формуле:

$$S = c * \sum_{i=1}^{m} \left(\sum_{j=1}^{l_i} v_i + p_i \right) \quad (3)$$

где: c - некоторый коэффициент пропорциональности, корректирующий взаимное использование метрик сложности вычислительных маршрутов и маршрутов принятия логических решений.

Критерии выделения маршрутов

Выделение маршрутов исполнения программы, которые необходимы для минимальной проверки всех возможных маршрутов ее выполнения и оценки структурной сложности, может осуществляться по различным критериям.

Наилучшим следует считать такой критерий, который позволит выделить все возможные маршруты исполнения программы при любых сочетаниях исходных и промежуточных данных. При этом формирование маршрутов зависит не только от структуры программы, но и от самих значений переменных в различные моменты времени и на различных участках выполнения программы. Такое выделение маршрутов трудно формализовать, и оно представляется весьма трудоемким для оценки показателей сложности программной структуры.

Рассмотрим более простые критерии выделения маршрутов, учитывающие только структурные характеристики программных модулей. При этом будем анализировать *граф потока управления,* под которым понимается множество всех возможных путей исполнения программы, представленное в виде графа. Следовательно, *граф потока управления* - ориентированный граф, моделирующий поток управления программой (часто граф потока управления именуется *управляющим графом). Поток управления* - последовательность выполнения различных модулей и операторов программы.

Приведенные критерии для оценки сложности программных модулей в каждом случае характеризуют минимально необходимые способы проверок. Для проверки реальных программ этого количества проверок может оказаться недостаточно (например, циклы целесообразно проверять не однократно, а на нескольких промежуточных значениях и, кроме того, на максимальном и минимальном количествах исполнения циклов). Оценить достаточность проверок программы значительно труднее, так как кроме сложности структуры при этом необходимо анализировать сложность преобразования каждой переменной во всем диапазоне ее изменения и при сочетаниях с другими переменными.

Критерий 1

Данный критерий предполагает, что граф потока управления программой должен быть проверен по минимальному набору маршрутов, проходящих через каждый оператор ветвления по каждой дуге.

Прохождение по каждому маршруту осуществляется только один раз, повторная проверка дуг не проводится и считается избыточной. При этом в процессе проверки гарантируется выполнение всех передач управления между операторами программы и каждого оператора не менее одного раза. Следует заметить, что в настоящее время существуют алгоритмы, позволяющие автоматизировать процесс получения минимального множества маршрутов именно по этому критерию.

Структурная сложность программы по первому критерию вычисляется следующим образом:

$$S_1 = \sum_{i=1}^{m} p_i \quad (4)$$

где p_i - количество вершин ветвления в *i-м* маршруте без учета последней вершины.

Критерий 2

Критерий основан на анализе базовых маршрутов в программе, которые формируются и оцениваются на основе *цикломатического числа* графа потока управления программы.

Цикломатическое число Z исходного графа потока управления программой определяется формулой

$$Z = m - n + 2 * p \quad (5)$$

где *m* - общее число дуг в графе; *n* - общее число вершин в графе; *p* - число связных компонентов графа.

Число связных компонентов графа p равно количеству дуг, необходимых для превращения исходного графа в максимально связный граф. *Максимально связным* (или *полносвязным*) *графом* называется такой граф потока управления, у которого любая вершина доступна из любой другой вершины. Максимально связный (полносвязный) граф может быть получен из исходного графа потока управления программой путем замыкания конечной и начальной вершин. Для большинства корректных графов достаточно одной замыкающей дуги, проведенной между начальной и конечной вершинами. Под корректными понимаются такие графы потоков управления, у которых не содержится висячих и тупиковых вершин.

Второй критерий, выбранный по цикломатическому числу, требует однократной проверки или тестирования каждого *линейно независимого цикла* и каждого *линейно независимого ациклического участка* (т. е. не имеющего в своем составе циклических участков) программы.

Каждый линейно независимый ациклический маршрут или цикл отличается от всех остальных хотя бы одной вершиной или дугой, т. е. его структура не может быть полностью образована компонентами, входящими в состав других маршрутов.

При использовании второго критерия количество проверяемых маршрутов равно цикломатическому числу.

Для *правильно структурированных* программ характерно отсутствие циклов с несколькими выходами, такие программы не имеют переходов внутрь циклов или условных операторов, не имеют принудительных выходов из внутренней части циклов или условных операторов. Для таких программ цикломатическое число Z можно определить путем подсчета числа вершин n_B, в которых происходит ветвление.

Тогда цикломатическое число можно представить следующим образом:

$$Z = n_B + 1 \quad (6)$$

Анализ графов реальных программных модулей с достаточно большим фиксированным числом вершин показывает следующее:

- суммарная сложность тестовых маршрутов почти не зависит от

детальной структуры графа и в основном определяется количеством ветвлений графа потока управления программы;

- при неизменном числе вершин в графах широкого профиля имеется большее количество маршрутов, чем в узких графах, но формируемые маршруты в среднем оказываются короткими.

Для автоматического анализа графов по второму критерию с помощью средств вычислительной техники используются матрицы смежности и достижимости графов, содержащие информацию о структуре проверяемой программы.

Матрица смежности - квадратная матрица, в которой единицы располагаются в позициях *(i,j)*, если в графе потока управления программой имеется дуга *(i,j).* В противном случае, при отсутствии дуги в такой позиции, ячейки матрицы просто не заполняются, обозначая нулевое значение в соответствующей позиции. Пример матрицы смежности показан в табл. 1 для управляющего графа, приведенного на рис. 1, доведенного до вида полносвязного графа (пунктирная линия).

Матрицей достижимости называется квадратная матрица, в которой единицы располагаются в позиции, соответствующей дуге *(i,j)*. Пример матрицы достижимости имеет вид, показанный в табл. 2.

Матрица достижимости является основным инструментом для вычисления маршрутов по второму критерию. Используя компьютер, матрицу достижимости можно получить из матрицы смежности путем возведения ее в степень, величина которой равна числу вершин без последней в исходном графе потока управления.

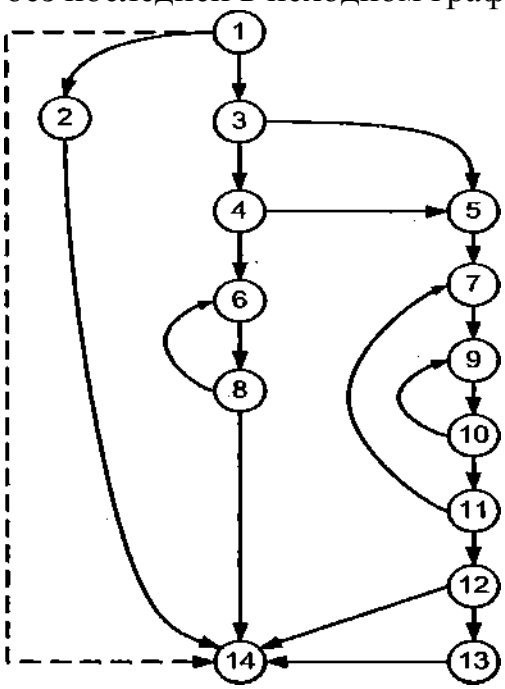

Рис. 1. Пример полносвязного управляющего графа

Таблица1. Матрица смежности

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1														
2	**1**													
3	**1**													
4			**1**											
5			**1**	**1**										
6				**1**				**1**						
7					**1**						**1**			
8						**1**								
9							**1**			**1**				
10									**1**					
11										**1**				
12											**1**			
13												**1**		
14		**1**						**1**				**1**	**1**	

Таблица2. Матрица достижимости

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1														
2	**1**													
3	**1**													
4	**1**		**1**											
5	**1**		**1**	**1**										
6	**1**		**1**	**1**		**1**		**1**						
7	**1**		**1**	**1**	**1**		**1**		**1**	**1**	**1**			
8	**1**		**1**	**1**		**1**		**1**						
9	**1**		**1**	**1**	**1**		**1**		**1**	**1**	**1**			
10	**1**		**1**	**1**	**1**		**1**		**1**	**1**	**1**			
11	**1**		**1**	**1**	**1**		**1**		**1**	**1**	**1**			
12	**1**		**1**	**1**	**1**		**1**		**1**	**1**	**1**			
13	**1**		**1**	**1**	**1**		**1**		**1**	**1**	**1**	**1**		
14	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	

С помощью матрицы достижимости можно сравнительно просто выделить циклы, отмечая диагональные элементы, равные единице, и идентичные строки. В табл. 2 для наглядности применена тонировка ячеек различными уровнями серого цвета.

Критерий 3

Более сильные критерии проверки и определения сложности структуры

программы включают требования однократной проверки не только линейно независимых, но и всех линейно зависимых циклов и ациклических маршрутов. К таким критериям относится третий критерий формирования маршрутов для тестирования программ.

Третий критерий выделения маршрутов основан на формировании полного состава базовых структур графа потока управления программой и заключается в анализе каждого из реальных ациклических маршрутов исходного графа программы и каждого цикла, достижимого из всех этих маршрутов. При этом каждый из указанных компонентов структуры программы должен быть проанализирован хотя бы один раз.

Если при прохождении какого-либо ациклического маршрута исходного графа потока управления достижимы несколько элементарных циклов, то при тестировании должны исполняться всё достижимые циклы.

Метрика Маккейба

Наиболее популярной метрикой, позволяющей оценить структурную сложность программных средств, является метрика Маккейба, построенная не на анализе лексических характеристик программы, а на результатах анализа потока управления от одного оператора к другому. Это позволяет (в отличие от метрик Холстеда) учесть логику построения программы при оценке ее сложности. На основе разработанных методов оценки сложности программ автор предложил стратегию проверки корректности ПС, которая получила название *основного маршрута тестирования* Маккейба.

Программное средство (алгоритм, спецификация) должно быть представлено в виде управляющего ориентированного графа

$$G = (V, E)$$

с V вершинами и E дугами, где вершины соответствуют операторам, а дуги характеризуют переход управления от одного оператора к другому.

Граф, описывающий программу в виде вершин-операторов и дуг-переходов, называют *графом потока управления* или *управляющим графом* программы.

Безусловно, для представления программы в виде графа необходимы определенные соглашения, определяющие положения о том, что считать узлом графа, так как синтаксис операторов в различных языках программирования может существенно отличаться. При этом обычно учитывают только исполнимые операторы и исключают группы операторов, назначением которых является описание данных. Желательно выбирать такие синтаксические формы операторов, которые в наибольшей степени подходят для отображения в виде узла графа.

Линейные участки программы вообще можно заменить одним узлом графа, относя к нему группы операторов, выполнение которых осуществляется в прямой последовательности без каких-либо условий (это характерно для участков программ, содержащих последовательность прямых вычислений переменных). В этом отношении использование алгоритмов в схемном представлении или описание программ на псевдокоде может оказаться более удобной формой описания программы, чем текст на языке программирования. При любом варианте представления анализируемой программы желательно преобразовать

операторы цикла в эквивалентную последовательность операторов ветвления, добавив, при необходимости, операторы подсчета числа повторений цикла с «верхним» или «нижним» окончанием (счетчики циклов).

Метрика Маккейба характеризует цикломатическое число графа потока управления программой и определяется следующим выражением:

$$M = m - n + 2 \qquad (7)$$

где *m* - количество ребер графа, *n* - число вершин графа.

Величину *M,* рассчитанную по этой формуле, называют *цикломатическим числом Маккейба.*

Цикломатическая сложность программы — структурная (или, иначе говоря, топологическая) мера сложности программы, применяемая для измерения качества ПО и основанная на методах статического анализа кода ПС. Цикломатическая сложность программы равна цикломатическому числу графа программы, увеличенному на единицу.

При вычислении цикломатической сложности используется граф потока управления программой: узлы графа соответствуют неделимым группам команд программы и ориентированным ребрам, каждый из которых соединяет два узла и соответствует двум командам, вторая из которых может быть выполнена сразу после первой. Цикломатическая сложность может также быть применена для отдельных функций, модулей, методов или классов в пределах анализируемого программного средства. Эта стратегия тестирования называется *основным маршрутом тестирования* Маккейба. Тестирование представляет собой проверку каждого линейного независимого маршрута графа программы. При такой проверке программы количество тестов должно быть равно ее цикломатической сложности.

Цикломатическая сложность части программного кода - счетное число линейно независимых маршрутов, проходящих через программный код. Например, если исходный код не содержит никаких точек решений, таких как указания условий IF или границы циклов (например, FOR), то сложность должна быть равна единице, поскольку есть только единственный маршрут выполнения программного кода. Если в тексте программы размещен единственный оператор IF, содержащий простое условие, то должно быть два пути выполнения кода: один путь через оператор IF с оценкой выполнения условия как ИСТИНА (TRUE) и другой - для альтернативной ветви при значении ЛОЖЬ (FALSE), в котором заданное условие не выполняется (рис. 1).

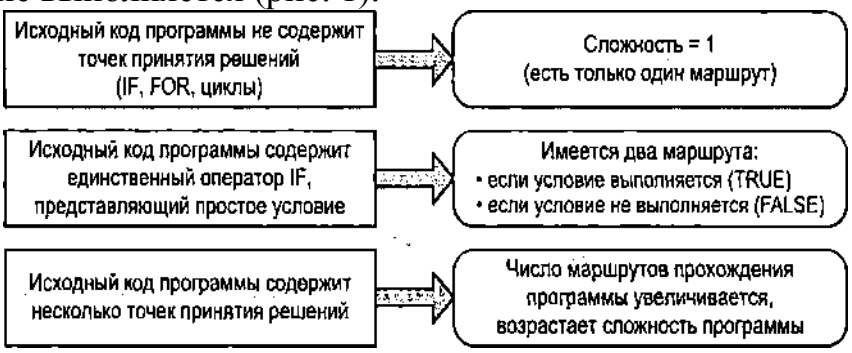

Рис. 2. Характеристика влияния точек принятия решений на сложность программы

Теоретической основой определения цикломатического числа Маккейба

является теория графов. В теории графов цикломатическое число ориентированного графа определяется выражением (5).

Число компонентов связности p можно рассматривать как минимально необходимое количество ребер, которые нужно добавить к графу, чтобы сделать его полносвязным. *Полносвязным* считается граф, у которого существует путь из любой вершины графа в любую другую вершину графа. Как показал анализ достаточно большого количества управляющих графов, для обеспечения полносвязности графа достаточно добавления одной фиктивной дуги (которая отсутствует в реальном графе программы, потому и считается фиктивной) из его конечной вершины в начальную. Поэтому можно считать, что практически для любого графа управления программой число компонентов связности равно единице, т. е. $p = 1$. Подстановка значения $p = 1$ в формулу определения цикломатического числа дает значение цикломатического числа Маккейба.

Цикломатическое число определяет количество независимых контуров в полносвязном графе и, как следствие, количество различных маршрутов, ведущих из начальной вершины в конечную.

При оценке сложности программы с применением цикломатического числа Маккейба справедливо следующее положение: если значение цикломатического числа больше 10, это означает, что программа обладает излишней сложностью и ее следует разбить на составные части (независимые модули или подпрограммы), для которых цикломатическое число будет иметь меньшее значение.

1.2. Особенности построения управляющих графов

Рассмотрим методику построения графов управления для типовых блоков, реализуемых основными операторами языков программирования. Напомним, что ориентированный управляющий граф позволяет учесть логику программы, обеспечивая возможность анализа потока передачи управления между операторами. Этот граф формируется из вершин, которые соответствуют операторам программы, и дуг, задающих переходы между этими операторами.

Прежде чем рассматривать типовые блоки программных модулей, отметим несколько общих замечаний, справедливых практически для любых программных блоков.

1. При построении графа управления программой надо учитывать только исполнимые операторы, ***не учитывая операторы описания***.
2. Если в программе существует несколько операторов, не изменяющих порядок действий в программе, и они следуют один за другим, их допускается ***объединять в одну вершину графа***.
3. При построении графа управления операторы цикла следует заменить несколькими вершинами. *Например, для оператора типа* for *должны быть вершины, соответствующие операторам начального присваивания счетчика цикла, его изменению и ветвлению, определяющим продолжение или завершение выполнения цикла.*
4. Аналогично несколькими вершинами заменяют и другие операторы цикла. Они должны соответствовать действиям, предшествующим циклу,

определяющим продолжение или прекращение выполнения цикла, а также группе операторов, составляющих тело цикла - множеству повторяемых действий. Таким образом, в графе должны быть три группы вершин, определяющих три элемента, составляющих любой цикл (начало цикла, тело цикла, ветвление при окончании цикла - завершение или возвращение к исходному оператору). *Далее это будет рассмотрено на конкретном примере.*

Правильно построенный управляющий граф позволяет оценить структурную (или топологическую) сложность программы, определяемую количеством независимых контуров в полносвязном графе.

Приведенные далее фрагменты исходных текстов программ, на основе которых будут рассматриваться особенности построения управляющих графов, написаны на разных языках программирования (C, C++ и C#), однако методика построения графов в равной степени применима к любым языкам программирования.

Линейная последовательность операторов

Рассмотрим фрагмент программы вычисления площади прямоугольника по длинам двух его сторон и приведем ее граф управления. Пример реализации такой программы на языке *C#* (приводятся только исполнимые операторы консольного приложения) приведен на рис. 3, а граф такого фрагмента показан на рис. 4.

Номера строк	Строки программы
1	**a=double.Parse(Console.ReadLine());**
2	**b=double.Parse(Console.ReadLineO);**
3	**p=a*b**
4	**Console.WriteLine("{0:f}", p);**

Рис. 3. Фрагмент программы вычисления площади прямоугольника

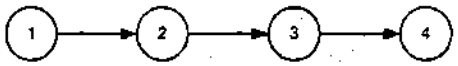

Рис. 4. Управляющий граф линейной последовательности операторов

Расчеты по такому графу управления программой будут такими:

- количество дуг $m = 3$,
- количество вершин $n = 4$,
- цикломатическое число Маккейба

$$Z = m - n + 2 = 3 - 4 + 2 = 1$$

Заметим, что текст программы снабжен номерами, которые соответствуют номерам вершин графа. *(Рекомендуется при самостоятельном выполнении практического занятия другие фрагменты программы снабжать такими номерами).*

Решение такой задачи можно записать по-другому, например так, как показано на рис. 5.

Номера строк	Строки программы
1	**a=double.Parse(Console.ReadLine());**
1	**b=double.Parse(Console.ReadLine());**
2	**Console.WriteLine("{0:f}", p=a*b);**

Рис. 5. Сокращенный вариант фрагмента программы

Для такой программы граф управления приведен на рис. 6.

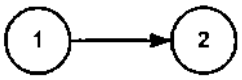

Рис. 6. Граф управления сокращенной линейной последовательностью операторов

Такой граф даст следующие результаты расчета характеристик:

- количество дуг: $m = 1$,
- количество вершин $n = 2$,
- цикломатическое число Маккейба

$$Z = m - n + 2 = 1 - 2 + 2 = 1.$$

Таким образом, цикломатическое число Маккейба осталось таким же, как и в первом случае. Этот пример подтверждает сделанное ранее утверждение о том, что последовательность выполняемых подряд операторов может быть объединена в одну вершину с одним входом и одним выходом, если они не изменяют последовательности выполнения программы.

Простое ветвление (оператор if)

Примеру программы с одним разветвлением может соответствовать задача вычисления максимального значения двух чисел и присвоения его результирующей переменной. В формализованном виде данную задачу можно записать следующим образом: *c = max {a, b}.*

Фрагмент реализации задачи на языке программирования С запишется, например, так, как показано на рис. 7.

Номера строк	Строки программы
1	**scanf(“%f %f”,&a,&Ь);**
2	**if (a>b)**
3	**c=a**
	else
4	**c=b;**
5	**prinrf(“%f”,c);**

Рис. 7. Фрагмент программы с разветвлением по оператору *if*

Граф управления для такого фрагмента приведен на рис. 8.

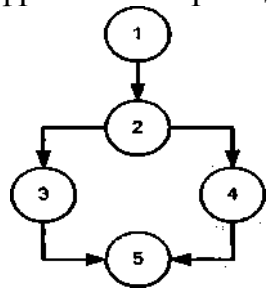

Рис. 8. Граф управления программы с разветвлением по оператору if

Характеристиками графа сложности такого фрагмента программы будут следующие показатели:

- количество дуг $m = 5$;
- количество вершин $n = 5$;
- цикломатическое число Маккейба

$$Z = m - n + 2 = 5 - 5 + 2 = 2.$$

Переключатель с множественным выбором

Такую функцию в программах реализуют операторы *switch*, *select*, *case* и им подобные.

Рассмотрим фрагмент программы, которая решает такую задачу.

В программу вводится символ. Необходимо определить, является ли такой символ допустимым для записи числа в римской системе счисления. Для упрощения примера будем распознавать не все символы, используемые в римской системе счисления, а примем к анализу только три: /, *V* и *X*. Фрагмент программы для решения такой задачи на языке C++ представлен на рис. 9.

Номера строк	Строки программы
1	**cout«"\n\n Input symbol I, V, X\n "; cin»s;**
2	**switch (s)**
	{
	case'I':
3	**cout«" I - good symbol "; break ;**
	case 'V':
4	**cout«" V - good symbol "; break;**
	case'X':
5	**cout«" X - good symbol "; break;**
	default:
6	**cout«" BAD SYMBOL ";**
	}
7	**getch();**

Рис. 9. Фрагмент программы с разветвлением по оператору *if*

Граф потока управления этим фрагментом программы приведен на рис. 10.

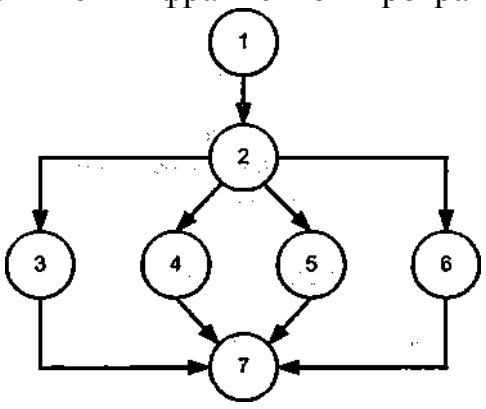

Рис. 10. Граф управления с множественным выбором

Количество дуг этого графа $m = 9$, а количество вершин $n = 7$. Рассчитаем для него цикломатическое число Маккейба:

$$Z = m - n + 2 = 9 - 7 + 2 = 4.$$

Отметим, что для тестирования такого фрагмента программы достаточно 4 теста (например, со следующими введенными значениями переменной *s: I, V, X,* v).

Заметим, что в графе на рис. 10 введена одна фиктивная вершина. Она соответствует участку программы, куда передается управление после завершения оператора *switch.*

Программы с операторами цикла

Пусть имеется программа, вычисляющая сумму первых *N* членов натурального ряда

$$S = \sum_{i=1}^{N} i.$$

Для построения графа потока управления при оценке структурной сложности программы был оператор цикла программы на эквивалентную последовательность операторов. Среди последних должны быть операторы присваивания и вычисления условий. Они должны отразить три элемента, присущих любому циклу:

- начальные присваивания до выполнения цикла;
- повторяющиеся действия (тело цикла);
- условия окончания (или продолжения) цикла.

Для представленного фрагмента программы это можно сделать, например, так, как показано на рис. 11, а управляющий граф представить в виде рис. 12.

Номера строк	Строки программы
1	**N = 5;**
2	**S = 0;**
3	**i = 1;**
4	**one: S + = i;**
5	**i=i+l;**
6	**if (i<= N) goto one;**
7	**Console.WriteLine("s = {0:d}", S);**

Рис. 11. Модифицированный фрагмент программы вычисления суммы членов натурального ряда

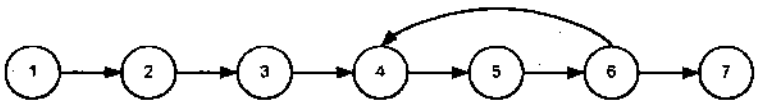

Рис. 12. Управляющий граф программы с циклом без объединения вершин

В графе вершины с первой по третью соответствуют действиям, предшествующим циклу, вершины три и четыре - группе операторов, составляющих тело цикла (множеству повторяемых действий), а вершина шесть - действию, определяющему необходимость продолжения или прекращения выполнения цикла. Расчет цикломатического числа Маккейба даст следующий результат:

$$Z = m - n + 2 = 5 - 5 + 2 = 2.$$

где количество дуг в графе равно $m = 5$, количество вершин $n = 5$.

Рассмотрим, как строится граф управления для программы с циклом *while.* Вновь обратимся к вычислению суммы первых членов натурального ряда, т. е. к задаче предыдущего примера. Фрагмент программной реализации задачи представлена на рис. 2.13.

Номера строк	Строки программы
1	**N = 5;**
2	**S = 0;**
3	**i = 1;**
4	**while (i<= N)**
5	**{**
6	**i++,.**
7	**S + = i;**
8	**}**
9	**Console.WriteLine("{0:d} ", S);**

Рис. 2.15. Фрагмент программы вычисления суммы членов натурального ряда с применением оператора *while*

Так же как и в предыдущем случае с циклом *do,* заменим оператор цикла эквивалентными операторами.

На рис. 14 приведены номера операторов до их объединения и соответствующие им номера после процедуры их приведения к одной вершине. Как уже объяснялось выше, объединять несколько операторов в группу и представить ее одной вершиной можно в том случае, если это последовательность линейных (следующих один за другим) операторов без ветвления процедуры выполнения программы. Управляющий граф такого фрагмента приведен на рис.15.

В этом графе управления количество дуг равно $m = 5$, количество вершин $n = 5$, цикломатическое число Маккейба

$$Z = m - n + 2 = 5 - 5 + 2 = 2.$$

Номера строк до объединения	Номера строк после объединения	Строки программы
1	1	N = 5;
2	1	S = 0;
3	1	i = 1;
4	2	one: if (i>= N) goto two
		{
5	3	i ++
6	3	S + = i;
7	4	goto one
		}
8	5	two: Console.WriteLine("{0:d} ", S);

Рис. 14. Фрагмент программы вычисления суммы членов натурального ряда с применением оператора *while*

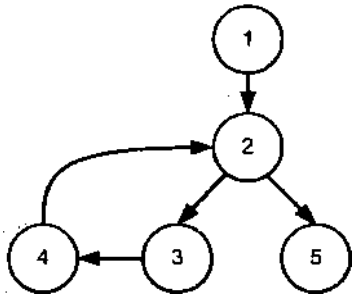

Рис. 15. Граф управления программой с циклом *while*

Как и для случая циклического оператора *do,* для отладки программы с оператором *while* может потребоваться два теста.

Теперь рассмотрим особенности построения графа управления для программы с циклом *for.* При этом предварительно вспомним, что циклы могут иметь две реализации: с предусловием и постусловием. Два описанных далее способа реализации цикла таковыми и являются. Оператор *do* реализуется с постусловием. В случае если количество повторений цикла задается таким, что цикл не должен выполниться ни разу, такой цикл даст неверный результат. Так, для рассмотренного ранее примера суммирования нескольких чисел натурального ряда при $N = 0$ результат (сумма) должна быть равна 0, в то время как программа сформирует результат 1. Случай же с предусловием (цикл типа *while)* даст ожидаемый результат, равный 0.

Еще одна особенность, которую следует отметить, состоит в том, что циклическая структура обязательно содержит три элемента: действия до начала цикла, тело цикла - повторяющиеся действия и условия окончания (или продолжения) цикла.

Первый из элементов всегда выполняется первым. Для циклов с предусловием вторым выполняется проверка, а затем тело цикла. Циклы с постусловием реализованы по-другому: сначала выполняется тело цикла, а затем осуществляется проверка. Если говорить о цикле *for,* то он может быть реализован как по схеме с предусловием, так и с постусловием. Для используемых средств реализации языков *C,* C# и *C++* оператор *for* реализован по схеме с предусловием и его граф управления программой похож на аналогичную структуру для оператора цикла *while.*

Рассмотрим более сложную задачу. Необходимо вычислить сумму факториалов заданного набора членов натурального ряда:

$$S = \sum_{i=1}^{n} i!$$

Для реализации решения задачи будем использовать цикл *for.*

Фрагмент программы решения этой задачи на языке C# может быть таким, как показано на рис. 16.

Следует обратить внимание, что в этой программе есть вложенный цикл.

Номера строк	Строки программы
1	**N = 5;**
2	**S = 0;**
3	**for (i=l; i<=N; i++)**

4	**{ f=1;**
5	**for (l=1; l<=l; 1++) f=f*l;**
6	**S + = i;**
7	**}**
8	**Console.WriteLine("{0:d} ", S);**

Рис. 16. Фрагмент программы вычисления суммы факториалов

Выполним, как и в предыдущем примере, замену операторов цикла и получим, например, такой текст программы (рис. 17).

Номера строк до объединения	Номера строк после объединения	Строки программы
1	**1**	**N = 5;**
2	**1**	**S = 0;**
3	**1**	**i=i;**
4	**2**	**onel: if (i>N) goto four;**
5	**3**	**f=1;**
6	**3**	**l=1;**
7	**4**	**two2: if (l>i) goto three;**
8	**5**	**f=f*l;**
9	**5**	**1++;**
10	**6**	**if (l<=i) goto two2;**
11	**7**	**three: S + = f;**
12	**7**	**i ++;**
13	**8**	**if (i<=N) goto onel;**
14	**9**	**four: Console.WriteLine("{0:d} ", S);**

Рис. 17. Фрагмент программы вычисления после замены операторов цикла

Теперь осуществим объединение линейных участков программы, которое приведет к тому тексту программы, на основе которого строится граф потока управления (рис. 18).

Граф потока управления для такой программы приведен на рис. 18, где группа вершин в тонированном контуре с пунктирными границами соответствуют вложенному циклу.

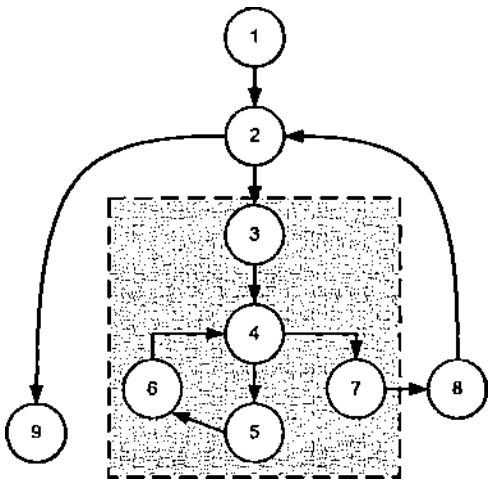

Рис. 18. Граф управления программой вычисления суммы факториалов

Цикломатическое число Маккейба для данного графа равно:

$$Z = m - n + 2 = 9 - 8 + 2 = 3.$$

Учитывая, что граф управления программой с циклом *for* может быть реализован также по схеме с постусловием, приведем пример построения графа управления и для этого случая. Заметим, что он будет похож на рассмотренный ранее граф с оператором *do.* (рассматривается задача вычисления суммы факториалов).

Произведем замену операторов цикла, тогда текст программы будет таким, как показано на рис. 19.

Номера строк до объединения	Номера строк после объединения	Строки программы
1	1	N = 5;
2	1	S = 0;
3	1	i=1;
4	2	one1: f=1;
5	3	l=1;
6	4	two2: f=f*1;
7	4	l++;
8	5	if (l<=i) goto two2;
9	6	S + = f;
10	6	i ++;
11	7	if (i<=n) goto onel;
12	8	Console.WriteLine("{0:d} ", S);

Рис. 19. Фрагмент программы вычисления для случая с постусловием

Управляющий граф для варианта использования схемы с постусловием приведен на рис. 20.

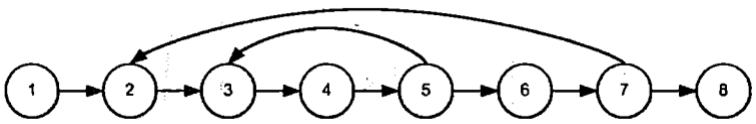

Рис. 20. Управляющий граф для схемы с постусловием

Пример решения задачи.

Постановка задачи:

Вычислить значение функции *G* = *F(x,* y), если

$$G = \begin{cases} true, \text{ если точка с координатами } (x, y) \text{ попадает в фигуру;} \\ false, \text{ если точка с координатами } (x, y) \text{ не попадает в фигуру.} \end{cases}$$

Фигура представляет собой сектор круга радиусом *R* = 2 в диапазоне углов 270° <*f* < 45°.

Разработать программу. В соответствии с разработанной программой составить блок-схему алгоритма решения задачи. На основании блок-схемы составить управляющий граф и оценить алгоритмическую сложность программы.

Решение задачи:

Текст программы для реализации возможного решения поставленной задачи, разработанной с использованием языка программирования C#, приведен на рис. 21.

Номера строк	Строки программы
1	using System;
2	class Operator
3	{
4	public static void Main()
5	{
6	const double R = 2.0;
7	double x, y; bool g;
8	char rep;
9	string str;
10	do
11	{
12	Console.Clear();
13	Console.Write("Введите X: ");
14	str = ConsoIe.ReadLine();
15	x = double.Parse(str); ,
16	Console. АУгйе("Введите Y: ");
17	str = Console.ReadLine();
18	y = double.Parse(str);
19	g = false;
20	if (x * x + y * y <= R * R)
21	if (x >= 0) ;
22	if (y <= x)
23	g = true;
24	str = string.Format("G({0:f3},{l: f3}) = {2}", x, y, g);
25	Console. WriteLine(str);
26	Console.Write ("Для повтора вычислений нажмите клавишу Y: ");
27	rep = char.Parse(Console.ReadLine());
28	Console. WriteLine();
29	} while (rep = 'Y' || rep = 'y');
30	}
31	}

Рис. 21. Программа расчета значения функции

Алгоритм решения задачи выглядит так, как показано на рис. 22.

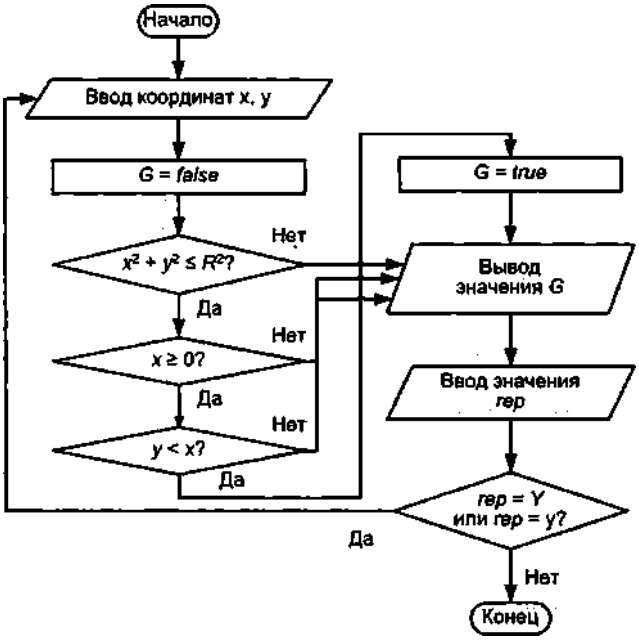

Рис. 22. Алгоритм вычисления значений функции

Оценка алгоритмической сложности

Граф потока управления для задачи расчета значений функции приведен на рис. 23. Тонированные вершины обозначают операторы ветвления.

Проведем оценку алгоритмической сложности графа по первому критерию. Определим минимальный набор маршрутов, проходящих через каждый оператор ветвления и по каждой дуге.

Для составленного управляющего графа можно выделить два маршрута:

m1: 1-2-**3**-**4**-**5**-6-7-8-**9**-2-**3**-7-8-**9**-10**;** p_1 **=6;**

m2: 1-2-**3**-**4**-7-8-**9**-2-**3**-**4**-**5**-7-8-**9**-10**;** $p_2 = 7$**.**

Данные маршруты проходят по всем вершинам ветвления, причем хотя бы один раз по каждой дуге графа, представленного на рис. 23, что соответствует требованиям первого критерия.

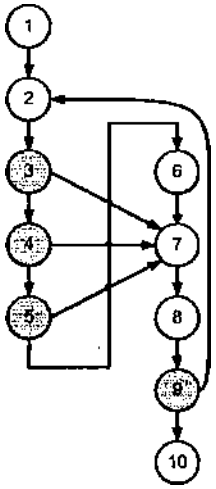

Рис. 23. Граф управления программой расчета значений функции

В перечне участков маршрутов номера вершин ветвления выделены полужирным шрифтом с подчеркиванием.

Таким образом, в соответствии **с первым критерием оценки алгоритмической сложности** необходимое количество маршрутов равно 2, количество вершин ветвления в маршрутах определяет уровень сложности:

$$S_1 = p_1 + p_2 = 6 + 7 = 13.$$

Проведем оценку алгоритмической сложности **по второму критерию**. Необходимо определить число проверок каждого линейно независимого цикла и линейно независимого ациклического участка программы. Количество проверок определяется цикломатическим числом графа, которое определяется следующим соотношением:

$$Z = n_B + 1,$$

где n_B-число вершин ветвления.

Число вершин ветвления в составленном графе составляет 4, отсюда

$$Z = n_B + 1 = 4 + 1 = 5.$$

Таким образом, общее число циклических и ациклических участков равно 5. Выделим маршруты на заданном графе:

- ациклические маршруты:

$m1$: 1–2–**3**–7–8–**9**–10; $p_1 = 2$;
$m2$: 1–2–**3**–**4**–7–8–**9**–10; $p_2 = 3$;
$m3$: 1–2–**3**–**4**–**5**–7–8–**9**–10; $p_3 = 4$;
$m4$: 1–2–**3**–**4**–**5**–6–7–8–**9**–10; $p_4 = 4$;

- циклические маршруты:

***m5:* 2-3-4-5-6-7-8-9; p_5=4.**

Тестирование программы по указанным маршрутам позволит проверить все операторы ветвления программы. Метрика структурной сложности определяется по следующему соотношению:

$$S_2 = p_1 + p_2 + p_3 + p_4 + p_5 = 2 + 3 + 4 + 4 + 4 = 17.$$

Для организации автоматического анализа заданного графа по второму критерию с помощью вычислительных средств построим матрицы смежности и достижимости.

Напомним, что матрица смежности представляет собой квадратную матрицу, размер которой определяется количеством вершин графа. Полученный граф имеет 10 вершин, следовательно, матрица смежности будет иметь размер 10×10. При заполнении матрицы следует придерживаться следующего правила: для дуги, соединяющей вершину 1 с вершиной 2, в первый столбец и вторую строку матрицы смежности записываем 1 (табл. 3). Номер столбца соответствует номеру вершины, из которой выходит дуга, а номер строки соответствует номеру вершины, в которую входит дуга. Аналогичным образом заполняем матрицу для всех дуг управляющего графа.

Таблица 3. Матрица смежности

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1										
2	**1**								**1**	
3		**1**								
4			**1**							
5				**1**						
6					**1**					
7			**1**	**1**	**1**	**1**				
8							**1**			
9								**1**		
10									**1**	

Для выделения маршрутов можно использовать матрицу достижимости, которая представляет собой для полученного графа управления квадратную таблицу размером 10x10. Номер столбца матрицы определяет номер вершины графа, из которого можно достичь другие вершины этого же графа, используя цикличные и ацикличные маршруты. Номера строк определяют номера достижимых вершин графа управления. Из вершины 1 возможно достичь вершины со второй по десятую, т. е. достижимы все вершины графа (см. рис. 23). Для первого столбца матрицы достижимости строки, начиная со второй, заполняются единицами (табл. 4). Далее аналогично заполняются столбцы для остальных вершин.

Таблица 4. Матрица достижимости

	1	2	3	4	5	6	7	8	9	10
1										
2	1	1	1	1	1	1	1	1	1	
3	1	1	1	1	1	1	1	1	1	
4	1	1	1	1	1	1	1	1	1	
5	1	1	1	1	1	1	1	1	1	
6	1	1	1	1	1	1	1	1	1	
7	1	1	1	1	1	1	1	1	1	
8	1	1	1	1	1	1	1	1	1	
9	1	1	1	1	1	1	1	1	1	
10	1	1	1	1	1	1	1	1	1	

Выделенные диагональные элементы матрицы определяют номера вершин, которые входят в состав циклических маршрутов. Идентичные строки матрицы определяют номера вершин, которые входят в состав ацикличных маршрутов.

Элементы, выделенные черным цветом, соответствуют маршруту *m*5: 2–**3**–**4**–**5**–6–7–8–**9**.

Все строки матрицы получились идентичными, так как все вершины графа входят в состав ацикличных маршрутов.

Проведем оценку алгоритмической сложности по третьему критерию. В соответствии с третьим критерием необходимо выделить все реально возможные маршруты управления:

*m*1: 1–2–**3**–7–8–**9**–10; $p_1 = 2$;
*m*2: 1–2–**3**–**4**–7–8–**9**–10; $p_2 = 3$;
*m*3: 1–2–**3**–**4**–**5**–7–8–**9**–10; $p_3 = 4$;
*m*4: 1–2–**3**–**4**–**5**–6–7–8–**9**–10; $p_4 = 4$;
*m*5: 1–2–**3**–**4**–**5**–6–7–8–**9**–2–**3**–**4**–**5**–6–7–8–**9**–10; $p_5 = 8$;
*m*6: 1–2–**3**–7–8–**9**–2–**3**–**4**–**5**–6–7–8–**9**–10; $p_6 = 6$;
*m*7: 1–2–**3**–**4**–7–8–**9**–2–**3**–**4**–**5**–6–7–8–**9**–10; $p_7 = 7$;
*m*8: 1–2–**3**–**4**–**5**–7–8–**9**–2–**3**–**4**–**5**–6–7–8–**9**–10; $p_8 = 8$.

Оценку структурной сложности программы рассчитываем по следующему соотношению:

$$S_3 = p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8 = 2 + 3 + 4 + 4 + 8 + 6 + 7 + 8 = 42.$$

Вывод:

Исходя из полученных результатов расчета метрик структурной сложности по первому ($S_1 = 8$), второму ($S_2 = 17$) и третьему ($S_3 = 42$) критериям выделения маршрутов можно сделать вывод, что программа, характеризуемая заданным графом управления, имеет невысокую алгоритмическую сложность, так как количество используемых в тексте операторов условий 5, для проверки которых необходимо проверить от 5 до 42 тестовых вариантов исходных данных.

Оценим алгоритмическую сложность программы **на основе метрики Маккейба.**

В соответствии с теорией Маккейба сложность алгоритма оценивается величиной цикломатического числа, которая определяется по следующему соотношению: $Z = m - n + 2$, где m - количество дуг управляющего графа, построенного на основе алгоритма программы, n - количество вершин графа. В соответствии с полученным управляющим графом $m = 13$, $n = 10$, тогда цикломатическое число равно:

$$Z = m - n + 2 = 13 - 10 + 2 = 5.$$

Вывод:

В соответствии со значением цикломатического числа ($Z = 5$) в полученном графе управления программой можно выделить пять независимых контуров, которые определяют пять управляющих маршрутов, ведущих из начальной вершины в конечную.

Значение цикломатического числа для полученного графа ($Z = 5$) не превышает значения 10, что говорит о незначительной сложности алгоритма решения задачи расчета значений функции.

2. Постановка задачи.

1) Каждым студентом самостоятельно выбирается программное средство, разработанное им на других лабораторных работах по любому предмету.

2) В соответствии с разработанной программой составить блок-схему алгоритма решения задачи.

3) На основании блок-схемы составить управляющий граф программы и оценить ее алгоритмическую сложность по каждому из критериев.

4) Сделать выводы алгоритмической сложности разработанной им программы.

3. Оформить отчет о проделанной работе.

Отчёт выполняется каждым студентом индивидуально. Работа должна быть оформлена в электронном виде в формате .doc и распечатана на листах формата А4. На титульном листе указываются: наименование учебного учреждения, наименование дисциплины, название и номер работы, вариант, выполнил: фамилия, имя, отчество, группа, проверил: преподаватель ФИО.

Отчет должен содержать:

- теоретические сведения;
- распечатку исходных данных (постановка задачи (для разрабатываемой программы), текст программы, блок-схема, управляющий граф, матрицы смежности и достижимости);
- используемые формульные соотношения;
- выводы по работе.