Практическое занятие №15

Тема: Оценка структурной сложности программ

Теоретические сведения

1.Критерии структурной сложности программ

1.1. 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 на 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 пре­ образования 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:

Sj (1)

где m - количество маршрутов исполнения программы; li – число данных, обрабатываемых в i-м маршруте; v1 - число значений обрабатываемых данных j-го типа (2 <= vj<= 5). ,

Расчет сложности программы по этой формуле имеет высокую неопределенность из-за отсутствия конкретных требований в выборе количества значений переменной vj 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 5-1О % ), 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)

где рi - 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, целесообразно установить способ определения общей сложности программы.

Общую сложность программы можно рассчитать по формуле

(3)

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

**Критерии выделения маршрутов**

Вычисление 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 по тока 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= (4)

где рi - 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 ·р, (5)

где m - общее число дуг в графе; п - общее число вершин в графе; р - число связных компонентов графа.

Число связных компонентов графа р 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 nв, в которых происходит ветвление.

Тогда цикломатическое число можно представить следующим образом:

Z=пв+ 1. (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 Таблице 1. для управляющего графа, приведенного на рис. 2.1, 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 Таблице 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.

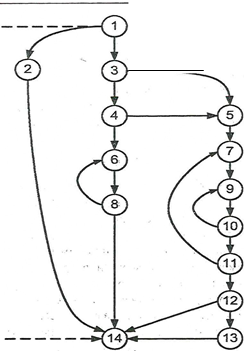


Рис. 2.1. Пример полносвязного управляющего графа

Таблица 1. 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 |  |

Таблица 2. 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 |  |  |  |  |  |  |
| 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. Для выделения вершин, входящих в состав различных циклов, использованы разные шрифты (полужирный и курсив).

**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 все 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 Е 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 исполняемые 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, (2.2.1)

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

Величину М, 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 (ТRUE) и другой - для альтернативной ветви при значении ЛОЖЬ (FALSE), в котором заданное условие не выполняется (рис. 2.1).

|  |  |  |
| --- | --- | --- |
| *Исходный код программы не содержит точек принятия решений*  *(IF, FOR, цикпы)* |  | *,,*  *Сложность = 1*  *(есть только один маршрут) ,,* |
|  |

|  |  |  |
| --- | --- | --- |
| Исходный ,код программы содержит . единственный оператор IF, представляющий простое условие | ,. | . Имеется два маршрута: . ·.· ,.•·   * если условие выполняется (ТRUE) * если условие не выполняется {FALSE) |
|  |

|  |  |  |
| --- | --- | --- |
| . *:.* .. -  Исходный код программы содержит несколько точек принятия решений |  | ··Число маршрутов прохождения ·  ·' ,. Программы увеличивается, возрастает сложность программы |
|  |

Рис; 2.2. 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

Z=m - n+2·р,

где m - количество ребер, n - число вершин, р - количество компонентов связности графа.

Число компонентов связности р 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. р = 1. Подстановка значения р = 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.

**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.

Отметим несколько общих замечаний, справедливых практически для любых программных блоков.

1. При построении графа управления программой надо учитывать только исполнимые операторы, не учитывать 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 fоr 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).

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

**Ответить на контрольные вопросы**

1.Чем определяется структурная сложность программы?

2. На какие группы можно разделить совокупность маршрутов исполнения программного модуля?

3.Для чего предназначена группа вычислительных маршрутов?

4.Что характерно для группы маршрутов принятия логических решений?

5. Что понимается под графом потока управления?

6. Что понимается под потоком управления?

7. Какие проверки предполагает Критерий 1?

8. На чем основан Критерий 2?

9. Что такое полносвязный граф?

10. Что понимается под корректным графом?

11. Чему равно количество проверяемых маршрутов при использовании Критерия2?

12.Что характерно для правильно структурированных программ?

13. Какие матрицы используются для автоматического анализа графов по Критерию 2 с помощью средств вычислительной техники?

14. На чем основан и в чем заключается Критерий 3 выделения маршрутов?

15.На чем основана, позволяющая оценить структурную сложность программных средств, метрика Маккейба?

16. Что характеризует метрика Маккейба и как она определяется?

17. Что называют цикломатическим числом Маккейба?

18. Что такое цикломатическая сложность программы?

19. Что определяет цикломатическое число?

20. При каком значении цикломатического числа программа обладает излишней сложностью?

21.Какие замечания следует учитывать при построении управляющих графов?