ГЛАВА ТЕОРЕТИЧЕСКАЯ

Целью данной главы является разработка модели сложности программы и комплексной меры, исследование и анализ методов расчета комплексной меры, исследование и анализ метрик программного обеспечения их достоинств и недостатков, исследование методов определения весовых коэффициентов на основе экспертных мнений и обоснование применения выбранного метода для расчета. Рассчитать весовые коэффициенты на основе выбранного метода, провести тестирование методики оценки сложности на контрольном примере.

## 2.1. Понятие сложности программы. Комплексная мера

На протяжении всего обучения необходимо оценивать уровень подготовленности обучающихся. В начале – для того, чтобы узнать начальный уровень, на последующих – чтобы наблюдать за прогрессом обучения. Чтобы определить этот уровень, необходимо знать с каким уровнем сложности задач может справиться студент на текущем этапе обучения. Для того чтобы решить задачу по программированию, необходимо разработать программу, которая должна на тестовый набор входных данных выдать соответствующий набор выходных данных удовлетворяющих условию задачи, поэтому для определения сложности задачи необходимо оценить сложность ее решения.

Зачастую под сложностью программы понимают различные понятия. Одним из них является сложность алгоритма. Трудоёмкость алгоритмов по-разному зависит от входных данных. Для некоторых алгоритмов трудоемкость зависит только от объема данных, для других алгоритмов — от значений данных, в некоторых случаях порядок поступления данных может влиять на трудоемкость. Трудоёмкость многих алгоритмов может в той или иной мере зависеть от всех перечисленных выше факторов .

Одним из упрощенных видов анализа, используемых на практике, является асимптотический анализ алгоритмов. Целью асимптотического анализа является сравнение затрат времени и других ресурсов различными алгоритмами, предназначенными для решения одной и той же задачи, при больших объемах входных данных. Используемая в асимптотическом анализе оценка функции трудоёмкости, называемая сложностью алгоритма, позволяет определить, как быстро растет трудоёмкость алгоритма с увеличением объема данных [15].

Иногда под сложностью понимают, насколько легко разобраться в программе, прочитав ее исходный код. Здесь оценивается стилистика и понятность и читабельность кода, которая завит от количества комментариев, поясняющих код, от разбиения больших участков кода на подпрограммы и.т.д. [13]

В данной работе под понятием сложности программы понимается следующее.

Сложность программы зависит от: размера программы; ее структур данных; структур управления, определяемых логикой программы; правильного ее разбиения на модули; внутренних связей каждого модуля; межмодульных связей в программе [18]. Таким образом сложность программы является латентной величиной, т.е. ее нельзя непосредственно измерить, но можно судить о значении сложности по определенным индикаторам ( свойствам программы ) – метрикам.

Так как о сложности программы приходится судить по ряду показателей - метрик, то для получения интегральной оценки воспользуемся комплексной мерой. Для расчета комплексной оценки используем формулу:

, (14)

где Mi – значения выбранных метрик, Ri ­­ - значения весовых коэффициентов.

Формула (14) строится на основе формулы Кокола [18]. Отличие заключается в том, что в первоначальном виде комплексная оценка строилась на базе одной главной метрики и корректировалась за счет некоторого количества дополнительных. В данной же формуле, отсутствует деление метрик на главные и корректирующие, а их влияние на итоговую оценку определяется соответствующим весовым коэффициентом, рассчитанным на основе экспертных мнений.

В итоге применение разработанной методики вычисления комплексной оценки производится в два этапа. Первый – это выбор метрик из уже существующих удовлетворяющих предметной области, и разработка собственных метрик для получения полноты данных о всех составляющих программы. Второй – расстановка весовых коэффициентов.

## 2.2. Анализ метрик сложности программы

На данном этапе определяются свойства программы, которые будут использованы для получения комплексной меры, и таким образом их можно измерить. Обзор метрик показал, насколько велико и разнообразно их количество. При выборе метрик необходимо учитывать ограничения, которые исходят из формы проведения практических занятий в форме контеста. Решение, подготовленное учащимся, представляет собой небольшую программу менее 500 строк кода. В программах нельзя использовать сторонние библиотеки. Исходный код программы должен быть в одном файле. Ввод данных производится из входного файла или со стандартного ввода.

Поэтому большинство метрик, используемых для оценок сложности, качества и надежности больших проектов, не будут эффективны для оценки программ обучающихся, исходя из этого необходимо исследовать существующие наборы метрик, которые оптимально подходят для предметной области.

### 2.2.1. Классификация метрик

Метрики сложности программ принято разделять на три основные группы [11]:

* объемные метрики;
* метрики сложности потока управления программ;
* метрики сложности потока данных программ.

К первой группе относятся метрики, основанные на количественной характеристике исходного кода программы. Эти метрики очень популярны, ввиду своей простоты. К наиболее известным метрикам данной группы относятся число операторов программы, количество строк исходного текста, набор метрик Холстеда[17]. Метрики этой группы ориентированы на анализ исходного текста программ.

Ко второй группе относятся метрики, основанные на анализе управляющего графа программы. Данный граф строится в виде ориентированного графа, в котором вершины – вычислительные операторы или выражения, а дуги – передача управления между узлами.

К третьей группе относятся метрики, основанные на оценке использования, конфигурации и размещения данных в программе. В первую очередь это касается глобальных переменных. К данной группе относятся метрики Чепина[19], подсчет спена[11].

**Метрические шкалы.**

В зависимости от характеристик и особенностей применяемых метрик им ставятся в соответствие различные измерительные шкалы.

Номинальной шкале соответствуют метрики, классифицирующие программы на типы по признаку наличия или отсутствия некоторой характеристики без учета градаций.

Порядковой шкале соответствуют метрики, позволяющие ранжировать некоторое характеристики путем сравнения с опорными значениями, т.е. измерение по этой шкале фактически определяет взаимное положение конкретных программ.

Интервальной шкале соответствуют метрики, которые показывают не только относительное положение программ, но и то, как далеко они отстоят друг от друга.

Относительной шкале соответствуют метрики, позволяющие не только расположить программы определенным образом и оценить их положение относительно друг друга, но и определить, как далеко оценки отстоят от границы, начиная с которой характеристика может быть измерена.

### 2.2.2. Метрика Холстеда

Метрика Холстеда относится к метрикам, вычисляемым на основании анализа числа строк и синтаксических элементов исходного кода программы.

Основу метрики Холстеда составляют четыре измеряемые характеристики программы:

* NUOprtr (Number of Unique Operators) — число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);
* NUOprnd (Number of Unique Operands) — число уникальных операндов программы (словарь операндов);
* Noprtr (Number of Operators) - общее число операторов в программе;
* Noprnd (Number of Operands) - общее число операндов в программе.
* На основании этих характеристик рассчитываются оценки:
* Словарь программы (Halstead Program Vocabulary, HPVoc): HPVoc = NUOprtr + NUOprnd;
* Длина программы (Halstead Program Length, HPLen): HPLen = Noprtr + Noprnd;
* Объем программы (Halstead Program Volume, HPVol): HPVol = HPLen log2 HPVoc;
* Сложность программы (Halstead Difficulty, HDiff): HDiff = (NUOprtr/2) × (NOprnd / NUOprnd);
* На основе показателя HDiff предлагается оценивать усилия программиста при разработке при помощи показателя HEff (Halstead Effort): HEff = HDiff × HPVol.

Данная метрика дает наиболее полную оценку о размере и сложности программы, основанную на анализе синтаксических элементов исходного кода программы.

На рисунках а) б) приведены примеры двух простейшие участков кода

void func()

{

cout << 1 << 2 << 3;

}

void func ()

{

cout << 1;

cout << 2;

cout << 3;

}

Если бы в качестве метрики сложности по размеру программы выступала LOC(lines of code ) то мы бы получили соответственно величины 2 и 4, хотя очевидно что эти два участка кода идентичны, за исключением того, что потребуется дополнительное время затратить на написание двух дополнительных раза «cout». Преимущество метрики Холстеда очевидно, так как и в первом и во втором случае получается одинаковое количество уникальных операторов 2, а количество уникальных операндов 4, общее число операндов в первом случае 4 во втором 6. Получаем сложность по Холстеду соответственно (15) и (16):

C\_H1 = 2/2 \* 4/4 = 1. (15)

C\_H2 = 2/2 \* 6/4 = 1.5. (16)

С другой стороны - одной метрики Холстеда не достаточно, так как эта метрика не исследует логику программы это видно на примере:

void func()

{

b = -a;

a -= c \* b;

c \*= a;

}

void func ()

{

if(a < b)

c = a + b;

else

c = a;

}

Участки кода имеют одинаковую сложность 7, так как имеют одинаковое число уникальных операторов 6, число уникальных операндов 3, число всех операндов 7.

C\_H = 6/2 \* 7/3 = 7. (17)

Однако, код с ветвлением должен обладать большей сложностью. Он более сложен в проектировании, более сложен в отладке и нужны дополнительные знания ЯП. Поэтому необходимо в совокупности метрики Холстеда использовать метрики анализирующие логическую структуру программы.

### 2.2.3. Метрика Мак-Кейба.

Относится к группе метрик сложности потока управления программ. Как правило, с помощью этих оценок оперируют либо плотностью управляющих переходов внутри программ, либо взаимосвязями этих переходов[3].

И в том, и в другом случае стало традиционным представление программ в виде управляющего ориентированного графа G=(V, E), где V – вершины, соответствующие операторам, а E – дуги, соответствующие переходам.

Показатель цикломатической сложности является одним из наиболее распространенных показателей оценки сложности программных проектов. Данный показатель был разработан ученым Мак-Кейбом в 1976 г., относится к группе показателей оценки сложности потока управления программой и вычисляется на основе графа управляющей логики программы (control flow graph). Данный граф строится в виде ориентированного графа, в котором вычислительные операторы или выражения представляются в виде узлов, а передача управления между узлами – в виде дуг.

Показатель цикломатической сложности позволяет не только произвести оценку трудоемкости реализации отдельных элементов программного проекта и скорректировать общие показатели оценки длительности и стоимость проекта, но и оценить связанные риски и принять необходимые управленческие решения.

Упрощенная формула вычисления цикломатической сложности представляется следующим образом:

C = e – n + 2, (18)

где e – число ребер, а n – число узлов на графе управляющей логики.

Как правило, при вычислении цикломатической сложности логические операторы не учитываются.

В процессе автоматизированного вычисления показателя цикломатической сложности, как правило, применяется упрощенный подход, в соответствии с которым построение графа не осуществляется, а вычисление показателя производится на основании подсчета числа операторов управляющей логики (if, switch и т.д.) и возможного количества путей исполнения программы.

Цикломатическое число Мак-Кейба показывает требуемое количество проходов для покрытия всех контуров сильносвязанного графа или количества тестовых прогонов программы, необходимых для исчерпывающего тестирования по принципу «работает каждая ветвь».

Показатель цикломатической сложности может быть рассчитан для модуля, метода и других структурных единиц программы.

Существует значительное количество модификаций показателя цикломатической сложности.

«Модифицированная» цикломатическая сложность – рассматривает не каждое ветвление оператора множественного выбора (switch), а весь оператор как единое целое.

«Строгая» цикломатическая сложность – включает логические операторы.

«Упрощенное» вычисление цикломатической сложности – предусматривает вычисление не на основе графа, а на основе подсчета управляющих операторов.

Данная метрика не может различать циклические конструкции и условные конструкции. Также недостатком метрики является невозможность оценки сложности предикатов, поэтому программы имеющие одинаковый граф будут иметь равную сложность, хотя будут иметь разные по сложности предикаты.

Рассмотрим следующие два примера с использованием конструкций if else и switch исходные коды и управляющие графы представлены на рисунке 2.

If(a == 1)

A();

else

if(a == 2)

B();

Else

C();

switch(a)

{

case 1: A(); break;

case 2: B(); break;

case 3: C(); break;

}

 

Рисунок 2. Управляющий граф программы примера 1(слева), примера 2 (справа)

Вычисли цикломатическую сложность для первого и второго примера (19) (20):

C\_M1 = 6 – 5 + 2 = 3. (19)

C\_M2 = 7 – 6 + 2 = 3. (20)

Таким образом, использование модифицированной цикломатической сложности, которая не учитывает каждое ветвление оператора switch, является не лучшим выбором для оценки решений, где каждый оператор имеет важное значение.

Существуют и очевидные минусы данной метрики в примере

while( a != 1)

A();

if(a == 1)

A();

else

B();

if(a == 1)

A();

Управляющие графы представлены на рисунке 3.



Рисунок 3. Управляющие графы программ

Цикломатическая сложность равна для всех 2, несмотря на то, что это три различные конструкции. К тому же при изменении условия в одном из примеров:

(a ==1) на (a == 1 && b!= 0 || b == 13) цикломатическая сложность не изменится. Таким образом, возникает необходимость для анализа составных условий, и четкое разделение конструкций на условные и циклические.

### 2.2.4. Метрика Джилба

Одной из наиболее простых, но, как показывает практика, достаточно эффективных оценок сложности программ является метрика Т. Джилба [7], в которой логическая сложность программы определяется как насыщенность программы выражениями типа IF-ELSE. При этом вводятся две характеристики: 1) CL – абсолютная сложность программы, характеризующаяся количеством операторов условия; 2) cl – относительная сложность программы, характеризующаяся насыщенностью программы операторами условия, т.е. cl определяется как отношение CL к общему числу операторов.

Дополнив метрику Джилба подсчетом циклических конструкций, можно получить метрику, которая позволит в совокупности с метрикой Мак-кейба эффективно определять цикломатическую сложность программы и распознавать наличие обеих ветвей условного оператора, а также анализировать циклические конструкций.

### 2.2.5. Дополнительные метрики

Для более полного анализа характеристик программ возникла необходимость в разработке дополнительных метрик. Данные метрики были разработаны в результате исследования особенностей спортивного программирования.

Метрика суммарной сложности условий. Целью данной метрики является расчет сложности составных условии в условном операторе if и циклических операторов for и while. Ни одна из выше описанных метрик не уделяет внимания на сложность условий. Однако чем больше составное условие тем сложнее отлаживать работу программы и зачастую в составных условиях допускаются ошибки.

Расчет метрики производится следующим образом. Для каждого составного условия подсчитывается количество логических операторов cntLO. Итоговое значение метрики получается по формуле :

где Cslo  - итоговое значение метрики, cntLOi – значение количество логических операторов i – го составного условия.

Для примера рассмотрим два условных оператора:

if(a == 1)

f();

if(!(fl && a == 2) )

f()

Для первого примера величина метрики равна Cslo  = 1 (один логический оператор “==”), для второго Cslo  = 3 (“!”, “&&”, “==”).

## 2.3. Расчет весовых коэффициентов

### 2.3.1. Прямая расстановка

Эксперты расставляют веса факторам, исходя из некоторого требования, например, чтобы сумма всех весов была равна единице либо 100%, или может быть выбрана и любая другая константа.

Трудности этого подхода заключаются в необходимости в неявном виде держать в поле зрения одновременно все факторы, поскольку, присваивая определенное числовое значение конкретному фактору, эксперт должен одновременно его сопоставить со всеми остальными. Сложности возрастают в геометрической прогрессии по мере увеличения числа факторов.

Есть еще и техническое затруднение в работе эксперта, связанное с необходимостью постоянно контролировать текущую сумму весовых коэффициентов, чтобы не оказаться перед фактом превышения заданной константы или оставить на последние факторы слишком большую часть. Если это происходит, то приходится переопределять уже присвоенные коэффициенты, что может происходить несколько раз, пока этот своеобразный итерационный процесс не закончится. Число итераций увеличивается по мере роста количества факторов.

### 2.3.2. Ранжирование факторов

Этот подход несколько облегчает экспертам работу, поскольку не требует контроля общей суммы коэффициентов. Здесь от экспертов требуется провести ранжирование, т. е. упорядочить обследуемые факторы, формирующие объект, по степени проявления их свойств в порядке их возрастания или убывания:

R11, R21, … , Rn1;

R12, R22, … , Rn2;

………………; (23)

R1m, R2m, … , Rnm

где *Rij* - ранг (место), присвоенный фактору *Oij* - м экспертом в ряду из *n* обследованных объектов, упорядоченных этим экспертом по степени проявления анализируемого свойства. Допускается двум и более факторам присваивать одинаковый ранг, но тогда он будет дробным. Сводные оценки весовых коэффициентов можно получить в результате усреднения частных рангов по столбцам.

Преимущество этого метода заключается в его простоте, но усреднение рангов ведет к более грубым оценкам весовых коэффициентов по сравнению с другими методами.

Кроме того, он также не избавляет эксперта от необходимости держать в поле зрения все факторы, как и при прямой расстановке.

### 2.3.3. Присвоение коэффициентов факторам*.*

В этом методе экспертам предлагается оценивать факторы по некоторой балльной шкале, например от 1 до 10. Тогда получаем выражения (24)

Y11, Y21, … , Yn1;

Y12, Y22, … , Yn2;

………………; (24)

Y1m, Y2m, … , Ynm

где *Yij* - балльная оценка фактора, полученная от *j*-го эксперта, *n* - количество факторов, *m* — число экспертов.

Сводные оценки весовых коэффициентов обычно находят путем подбора соответствующей регрессионной модели. Среднюю оценку *wi* весовых коэффициентов факторов можно получить по простым формулам

где *wij -* вес *i-*го объекта, рассчитанный по оценкам всех экспертов;

где *xij -* оценка фактора *i*, данная экспертом *j*; *n -* число факторов, *m* - число

экспертов.

Этот метод в определенной степени делает более слабой зависимость оценки конкретного фактора от остальных, но окончательно не избавляет от нее, поскольку сопоставлять факторы все же требуется – иначе коэффициенты значимости корректно невозможно расставить.

### 2.3.4. Метод анализа иерархий

Частично избавиться от указанных выше сложностей призван метод анализа иерархий (МАИ) [16], разработанный Т. Саати. Суть метода заключается в следующем. Факторы сравниваются между собой по парам относительно друг друга по их влиянию на конечную цель. При этом влияние других факторов не учитывается. Для попарного сравнения факторов автором метода Саати предложена специальная оценочная шкала, состоящая из пяти основных и четырех промежуточных суждений. В ней суждения экспертов представляются следующим образом (табл. 2):

Таблица 2. Иерархия экспертных сравнений соотношения факторов

|  |  |
| --- | --- |
| Суждение | Пояснение |
| 1. Равная важность | Равный вклад факторов в цель |
| 1. … | Промежуточное суждение |
| 3. Умеренное превосходство | Опыт и суждение дают легкое превосходство одного |
| 4. … | Промежуточное суждение |
| 5. Существенное превосходство | Сильное превосходство одного фактора над другим |
| 6. … | Промежуточное суждение |
| 7. Значительное превосходство | Имеется практически значительное превосходство одного фактора над другим |
| 8. … | Промежуточное суждение |
| 9.Очень сильное превосходство | Имеется значительное превосходство одного фактора над другим |

В итоге результаты парных сравнений представляются в виде квадратной матрицы A = (*aij*) с единичной диагональю (сравнение фактора самого с собой равно единице). Здесь *aij* означает отношение оценок соответствующих элементов; индексы *i* и *j* изменяются от единицы до величины, равной количеству факторов. Поскольку при последовательном переборе всех возможных пар факторы сравниваются между собой дважды (сначала - фактор *ai* с фактором *aj*, затем - в обратном порядке), при составлении матрицы должно выполняться условие «обратной симметричности»:

. (25)

Из условия (25) следует, что достаточно заполнять только одну часть матрицы, лежащую выше или ниже диагонали, что не имеет принципиального значения вследствие элементарного пересчета взаимно обратных значений. Если рассматривается *n* факторов, то всего возможно наличие значащих сочетаний.

В МАИ для кодирования используется номер соответствующей строки табл. 1. Каждое из приведенных суждений кодируется числом от 1/9 до 9. Например, если придано существенное превосходство элемента *Ai* над элементом *Аj*, то полагают, что в матрице парных сравнений *aij* = 5 и соответственно *aji* = 1/5, поскольку для кодирования используется пятая строка.

Суть обработки матрицы заключается в разложении: *A* ≈ *Z·U*, где . Цель - определение компонент вектора весов Z = (z1, …, z*n*),

что позволяет ранжировать факторы *Ai*.

Вычисление весов можно осуществить несколькими способами. Одним из возможных подходов к аппроксимации вектора весов может служить путь вычисления собственного вектора матрицы парных сравнений, который равен соответствующему максимальному собственному числу.

Положительные стороны метода. Отсутствие необходимости постоянно держать в поле зрения все факторы или, по крайней мере, группу однородных факторов, позволяет эксперту сконцентрировать внимание на конкретной проблеме: на сколько фактор *Аi* превосходит фактор *Bj* или уступает ему. Вследствие этого следует ожидать более точных результатов.

В практике нередко возникают ситуации, когда число влияющих факторов изменяется. В МАИ это приводит только к необходимости сравнения вновь возникших пар или же к вычеркиванию строк и столбцов матрицы парных сравнений, соответствующих изъятых из рассмотрения факторов, т. е. к образованию минора матрицы. Полученные результаты предыдущих опросов сохраняются, и полного обновления анкеты, как это происходит в других случаях, не требуется. С учетом того, что процедура МАИ, в сущности, сводится к поиску собственного вектора соответствующей матрицы, принадлежащего максимальному собственному значению, с «технической» точки зрения включение дополнительных факторов есть увеличение размерности соответствующего линейного пространства за счет добавления прямых слагаемых.

Обычные числовые шкалы не всегда удобны для сопоставления факторов, выражаемых в различных размерностях и понятиях. Особенно сложно сравнивать факторы, показателями которых, с одной стороны, являются количественные величины, а с другой – качественные. Вербально-числовые шкалы, одним из вариантов которых является шкала Саати, как раз и призваны оценивать такие несоответствия показателей влияющих факторов.

**Недостатки метода.** Возникает ряд вопросов при интерпретации результатов, и связаны они, прежде всего, с критерием качества работы эксперта — с отношением согласованности***.*** Отношение транзитивности хорошо работает, когда все характеристики исследуемой системы можно представить числовыми величинами. Но как только это становится невозможным, требование наличия транзитивности зачастую вступает в противоречие с логикой исследователя.

### 2.3.5. Пример расчета весовых коэффициентов по МАИ

Рассмотрим процесс расчета весовых коэффициентов на основе метода анализа иерархий. Эксперт попарно сравнивает влияние метрик на итоговую оценку. На основе опроса эксперта строим матрицу парных сравнений. Результаты представлены в таблице 3.

Таблица 3. Матрица парных сравнений

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Метрика Холстеда | Метрика Мак-Кейба | Метрика Джилба | Суммарная сложн. условий | кол-во рекурс.  вызовов | суммарная вложенность |
|  | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 1 | 1/5 | 1/3 | 1/2 | 1/2 | 1/2 |
| 2 | 5 | 1 | 2 | 3 | 3 | 3 |
| 3 | 3 | 1/2 | 1 | 2 | 2 | 2 |
| 4 | 2 | 1/3 | 1/2 | 1 | 1 | 1 |
| 5 | 2 | 1/3 | 1/2 | 1 | 1 | 1 |
| 6 | 2 | 1/3 | 1/2 | 1 | 1 | 1 |

Необходимо найти максимальное собственное число матрицы и соответствующий ему собственный вектор.

Решение несимметричной задачи собственных значений осуществляется в несколько этапов. На первом этапе матрица приводится ортогональным преобразованием к верхней форме Хессенберга. На втором этапе, занимающем больше всего времени, матрица приводится ортогональным преобразованием к верхней форме Шура. Если требуются только собственные значения, то этого достаточно, т.к. собственные числа матрицы располагаются в диагональных блоках квазитреугольной матрицы из канонической формы Шура. Если же требуются собственные векторы, то они могут быть получены по векторам Шура и квазитреугольной матрице путем обратной подстановки (фактически - решения системы линейных уравнений; сам процесс обратной подстановки занимает незначительную часть времени работы алгоритма, но необходимость накапливать проводимые над матрицей преобразования для применения их к собственным векторам замедляет алгоритм более чем в два раза).

Разложение Шура, на которое тратится больше всего времени, осуществляется с использованием QR-алгоритма. Более подробно нахождение в монографии [6].

Для нахождения собственных значений можно воспользоваться готовыми решениями для их расчета, например: библиотека lapack, пакет MathCad.

В итоге получаем (26):

x = . (26)

## 2.4. Пример расчета сложности программы

Для примера произведена оценка сложности следующей задачи.

**Условие.**

Дан массив. Нужно найти количество элементов, которые меньше среднего значения элементов массива.

<Input>

Со стандартного устройства ввода вводится целое число N – количество элементов массива, в следующей строке – N целых чисел – элементы массива.

<Output>

Нужно выдать на стандартное устройство вывода целое число – количество искомых элементов.

<Sample Input>

5

2 4 6 10 8

<Sample Output>

2

**Авторское решение.**

#include <stdio.h>

int main()

{

int n[1000],i,s,r;

scanf("%d",&n[0]);

s=0;

for(i=1;i<=n[0];i++)

{

scanf("%d",&n[i]);

s=s+n[i];

}

if(s%n[0]==0)

s=s/n[0];

else

s=s/n[0]+1;

r=0;

i=1;

while(i-1!=n[0])

{

if(n[i]<s)

r++;

i++;

}

printf("%d",r);

return 0;

}

Для начала рассчитаем метрики Холстеда:

* Количество уникальных операндов NUOprnd = 11
* Количество уникальных операторов NUOprtr = 20
* Общее число операндов NOprnd = 64
* Общее число операторов NOprtr = 57

Размерная сложность по Холстеду равна (27):

HDiff = (NUOprtr/2) × (NOprnd / NUOprnd) = 50. (27)

Далее построим управляющий граф программы, который представлен на рисунке 4.



Рисунок 4. Управляющий граф программы

По управляющему графу программы рассчитываем метрику Мак – Кейба. Количество вершин графа v = 11, количество ребер e = 14. Цикломатическая сложность по Мак – Кейбу (28):

C = e – v + 2 = 14 -11+2 = 5. (28)

Далее рассчитаем оставшиеся метрики, значение всех метрик представлено в таблице 4

Таблица 4. Значения метрик для тестового примера

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метрика Холстеда | Метрика Мак-Кейба | Метрика Джилба | Суммарная сложн.условий | Кол-во рекурс. вызовов | Сумм. уровень вложенности |
| 50 | 5 | 5 | 3 | 0 | 3 |

Используя формулу (14) и рассчитанные коэффициенты (26) получаем:

H\_M = 50\*1.1 + 5\*6.1 + 5\*3.7 + 3\*2 + 0\*2 + 3\*2 = 116 (28)

## 2.5. Выводы

Определены понятия сложности программы и комплексной меры. Проведен анализ методов расчета комплексной меры. Произведен анализ и исследование метрик программного обеспечения их достоинства и недостатки, представлены примеры расчета метрик. Разработаны дополнительные метрики программы и методы их расчета для более полного анализа сложности программы. Исследованы методы определения весовых коэффициентов на основе экспертных мнений и обосновано применение метода МАИ для расчета весовых коэффициентов, приведен пример расчета весовых коэффициентов для 6 метрик на основе метода МАИ. Приведены результаты тестирования методики оценки сложности на контрольном примере.