Глава 1. АЛГОРИТМЫ И АВТОМАТЫ

1.1. Общие понятия

Предшественниками вычислительных машин являются автоматы – устройства, выполняющие некоторый процесс без участия человека. Появление автоматов относится к глубокой древности. Это были различные механические игрушки, автоматы для открывания дверей в храмах и т. п. Одно из отличий вычислительных устройств наших дней от традиционных средств состоит в том, что они стали машинами, пригодными для автоматизации умственной деятельности человека.

Необходимость формального описания компьютера и его отдельных частей в процессе его проектирования требует применения специального математического аппарата, который необходим при любых разработках различных методов обработки информации, при синтезе и анализе любых информационных процессов. Для этого вводится понятие абстрактного (т. е. существующего не реально, а лишь в воображении) цифрового автомата.

Теория автоматов вошла в обиход в 50-е годы XX столетия, хотя соответствующая проблематика начала складываться еще в тридцатые годы в рамках теории автоматов и релейных устройств. Тогда в теории алгоритмов были сформулированы понятия вычислительного автомата (машины Тьюринга и Поста). Было установлено, что для осуществления всевозможных преобразований информации вовсе не обязательно сгроить каждый раз специализированные автоматы: все это можно сделать на одном универсальном автомате при помощи подходящей программы и соответствующего кодирования. Этот теоретический результат позже получил инженерное воплощение в виде современных универсальных вычислительных машин.

Обычно вычисления или машинные преобразования, связанные с решением задачи, представляют собой многошаговый процесс последовательных действий. До начала вычислений надо выбрать метод решения задачи и иметь предписание об операциях над исходными данными и порядке их выполнения. Предписание, определяющее порядок выполнения операций над данными с целью получения искомого результата, в нестрогом понимании называется алгоритмом.

Процесс подготовки решения задачи на ЭВМ часто называют алгоритмизацией. Обычно под алгоритмизацией процесса решения некоторого класса задач понимают описание последовательности действий, которые необходимо выполнить, чтобы задать процесс в виде однозначно определенной цепочки операций на языке математических символов. Чаще всего приходится начинать его со словесной формулировки при постановке задачи.

Затем с помощью формул, таблиц, схем и т. п. описывается метод решения задачи в соответствии с предлагаемым методом выбирается алгоритм решения.

Естественно, что понятие алгоритма в теории алгоритмизации занимает центральное место. Над уточнением понятия «алгоритм» работали многие ученые. На основе этих работ развилась теория вычислимых функций, конечных и бесконечных автоматов, являющихся математическими моделями вычислительных машин.

Вычислительная машина, вообще говоря, перерабатывает входную последовательность знаков в выходную. Операции нал этими последовательностями практически сводятся к применению ряда подстановок, которые позволяют выполнять различные преобразования: кодирование информации в вид, удобный для восприятия машиной; переработку информации по некоторому алгоритму; воспроизведение результата в удобной для человека форме.

При переработке информации рассматриваются конечные и бесконечные последовательности различных символов (букв). Конечные последовательности легко обозримы, но что значит «рассмотреть» бесконечную последовательность?

Представление о такой последовательности можно составить лишь по описанию тех или иных ее свойств. Например, определить по номеру места символ в бесконечной последовательности 01 01 01 ... . Если рассматривать только начало последовательность, то никаких представлений о ней в целом сделать нельзя. Однако такое представление создается, если указывается, что 0 и 1 чередуются всегда. Теперь можно предсказать, какой символ, например, стоит на 93-м и 1026-м месте.

В этом примере свойство, благодаря которому имелась возможность составить представление обо всей бесконечной последовательности, заключалось в существовании процедуры, позволяющей узнавать по номеру места цифру, стоящую на этом месте. Точнее можно сказать, что для этого примера имелся алгоритм распознавания символа по номеру места.

Интуитивное определение алгоритма. Существует несколько определений термина «алгоритм». Вообще под алгоритмом часто понимается некоторое формальное предписание, действуя согласно которому, можно получить нужное решение задачи. Эта формулировка соответствует интуитивной точке зрения на алгоритм, сложившейся еще в древности.

Классическим примером является алгоритм Эвклида для нахождения наибольшего общего делителя положительных чисел *а* и *b*.

1. Обозревай данные числа а и b. Переходи к указанию 2 (ПКУ2).
2. Сравни обозреваемые числа (***а*** = b или a < b или a > b). ПКУ З.
3. Если обозреваемые числа равны, то каждое из них дает искомый результат и процесс вычислений останавливается, иначе ПКУ4.
4. Если первое обозреваемое число меньше второго, то переставить их местами. ПКУ 5.
5. Вычитай второе число из первого и обозревай два числа: вычитаемое и остаток. ПКУ 2 (процесс повторяется по вышеназванной схеме).

Рассмотрим пример, иллюстрирующий применение алгоритма Евклида для чисел a = 15 и b = 20, указывая результаты и номера выполняемых операций.

По определению акад. А.Н. Колмогорова, алгоритм или алгорифм – это всякая система вычислений, выполняемых но строго определенным правилам, которая после какого-либо числа шагов заведомо приводит к решению поставленной задачи. В инженерной практике часто используется следующее определение: алгоритм конечная совокупность точно сформулированных правил решения какой-то задачи.

Алгоритмы, в соответствии с которыми решение поставленных задач сводится к арифметическим действиям, называются численными. Численными являются также любые алгоритмы для решения некоторого класса задач, если формулами полностью описан как состав действий, так и порядок их выполнения.

Указания, задающие элементарные действия, называются операторами. Реализация оператора сводится к переработке некоторой информации и к определению оператора, выполняемого вслед за данным.

Общие свойства алгоритмов. Из рассмотренных примеров отчетливо выступают свойства, присущие любому алгоритму.

Дискретность алгоритма. Алгоритм – процесс последовательного построения величин, идущих в дискретном времени таким образом, что в начальный момент задается исходная конечная система величин, а в каждый следующий момент система величин строится по определенному закону из системы величин, имевшихся в предыдущий момент времени.

Детерминированность алгоритма. Система величин, получаемых в какой-то не начальный момент времени, однозначно определяется системой величин, полученных в предыдущие моменты времени.

Элементарность шагов алгоритма. Закон получения последующей системы величин из предыдущей должен быть простым и локальным.

Направленность алгоритма. Если способ получения последующей величины из предыдущей не дает результата, должно быть указано, что считать результатом.

Массовость алгоритма. Начальная система величин может выбираться из некоторого бесконечного множества. Иными словами, алгоритм служит для решения целого класса задач.

Область применения. Областью применения алгоритма называется такая наибольшая область начальных данных, на которой алгоритм результативен.

Относительно перечисленных свойств алгоритма укажем их интерпретацию для алгоритма Евклида:

Дискретность. В любой момент по паре чисел (а, b) строится новая пара (а, b).

Детерминированность. Пара (а, b) определяется однозначно.

Элементарность шагов. Вычитание двух чисел, сравнение, перестановка.

Направленность. Указано правило прекращения процесса и то, что считается результатом выполнения каждого шага.

Массовость. Алгоритм применим к любой паре целых чисел а > О, Ь > 0.

Область применения, a,b ∈ {1,2,...}.

Следует отметить, что число операций при выполнении того или иного алгоритма заранее неизвестно и зависит от выбора исходных данных. Поэтому под осуществимостью алгоритма следует понимать потенциально возможный процесс для конкретных задач из области его применения, так как для решения некоторых из них может не хватить ни времени, ни памяти.

Математическое определение алгоритма. Алгоритмами в современной математике принято называть конструктивно задаваемые соответствия между изображениями объектов в абстрактных алфавитах.

Абстрактным алфавитом называется любая конечная совокупность объектов, называемых буквами или символами данного алфавита. При этом природа этих объектов нас совершенно не интересует. Символом абстрактных алфавитов можно считать буквы алфавита какого-либо языка, цифры, любые значки и даже слова некоторого конкретного языка. Основным требованием к алфавиту является его конечность. Алфавит, как любое множество, задается перечислением его элементов.

Итак, объекты реального мира можно изображать словами в различных алфавитах. Это позволяет считать, что объектами работы алгоритмов могут быть только слова. Тогда можно сформулировать следующее определение.

Алгоритм есть четкая конечная система правил для преобразования слов из некоторого алфавита в слова из этого же алфавита.

Слово, к которому применяется алгоритм, называется входным. Слово, вырабатываемое в результате применения алгоритма, называется выходным. Совокупность слов, к которым применим данный алгоритм, называется областью применимости этого алгоритма.

Можно выделить три основных типа универсальных алгоритмических моделей, различающихся исходными эвристическими соображениями относительно того, что такое алгоритм. Первый тип основан на функциональном подходе и рассматривает понятие алгоритма с точки зрения того, что можно вычислить с его помощью. Наиболее развитая и изученная модель этого типа – рекурсивная функция – является исторически первой формализацией понятия алгоритма.

Второй тип основан на представлении алгоритма как некоторого детерминированного устройства, способного выполнять в каждый отдельный момент некоторые примитивные операции, или инструкции. Основной теоретической моделью этого типа, созданной в 30-х годах, является машина Тьюринга, которая представляет собой автоматную модель, в основе которой лежит анализ процесса выполнения алгоритма как совокупности набора инструкций.

Третий тип алгоритмических моделей – это преобразования слов в произвольных алфавитах, в которых элементарными операциями являются подстановки, т. е. замены части слова (полслова) другим словом. Примерами моделей этого типа являются нормальные алгоритмы Маркова и канонические системы Поста.

Параллельный алгоритм. Параллельный алгоритм – алгоритм, операции которого могут выполняться одновременно. Приведенные выше определения алгоритма рассматривают его как процесс последовательной обработки структур данных, протекающий в дискретном времени так, что в каждый следующий момент времени структура данных получается по определенным правилам по структуре данных величин, имевшихся в предыдущий момент. Структура элементарных шагов в определении алгоритма не оговаривается, поэтому, правила преобразования величин могут допускать или предписывать параллельную их обработку. Алгоритм может быть параллельным, когда некоторые шаги обработки выполняются параллельно. Так, алгоритм сложения векторов может быть представлен как последовательный, в виде: *Сi* = Аi + Вi для i = 1,2, …, *n*, или в матричной записи: С = А + В, семантика которого попускает параллельную обработку элементов векторов.

Если при решении некоторой задачи процессоры выполняют одинаковую последовательность вычислений, но используют разные данные, то говорят о параллелизме по данным. Например, при поиске по базе данных каждый процессор может работать со своей частью базы данных. Если процессоры выполняют разные задания одной задачи, выполняют разные функции, то говорят о функциональном параллелизме.

Основная цель параллельных вычислений – уменьшение времени решения задачи. Чтобы ускорить решение задачи, не достаточно иметь параллельную вычислительную систему. Кроме этого, нужно ещё создать для такой системы специальную (параллельную) программу. Для того чтобы алгоритм мог быть эффективно реализован в виде параллельной программы, он должен обладать внутренним параллелизмом (алгоритм можно разбить на параллельные, но не обязательно независимо выполняемые части). Распараллеливание, осуществляемое до начала выполнения алгоритма, называют статическим, а осуществляемое во время выполнения алгоритма динамическим.

С параллельностью связано понятие масштабируемости. Масштабируемая параллельная система – это такая параллельная система, производительность которой пропорциональна числу содержащихся в ней процессоров. Масштабируемость зависит от возможностей коммуникационных сетей. Масштабируемость зависит также от параллельного алгоритма: алгоритм, проверенный и хорошо работающий на вычислительной системе (ВС) с малым числом процессоров может плохо работать (не давать ожидаемого ускорения) на системе с большим числом процессоров.

Рассмотрим, например, алгоритм вычисления выражения вначале нужно вычислить произведения *а*×*b* и *с*×*d*, затем извлечь корень, и, наконец, выполнить сложение. Описанный алгоритм можно изобразить следующим образом:

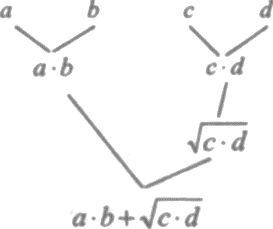


Схема параллельного алгоритма

Мы видим, что процесс обработки данных может быть выражен в виде однонаправленного графа. Такой граф можно изобразить на плоскости, причем каждую арифметическую операцию располагать максимально высоко (если ось времени направлена вниз), но не выше тех операций, результат которых нужен для её вычисления. В таком случае высота графа будет равна минимальному времени (числу шагов) решения этой задачи на идеальной параллельной ВС с неограниченным числом вычислителей.

Степень параллелизма (параллельная сложность) алгоритма называется число его операций, которые можно выполнить параллельно (ширина графа, построенного описанным выше способом). Степень параллелизма алгоритма сложения векторов равна *п*. Операции, которые всегда могут быть выполнены на идеальной ЭВМ параллельно независимо от их количества, иногда называют операциями фиксированной глубины.

1.2. Описание алгоритмов

Всякий алгоритм, как уже говорилось ранее, отражает последовательность преобразований информации. Алгоритм обычно состоит из операционной и информационной частей.

Естественно, что при описании алгоритмов нужно задавать операторы и связи между ними. Как правило, оператор представляет собой довольно простую конструкцию, и поэтому наибольшие трудности при разработке алгоритма приходятся на указание связей между операторами, т. е. на описание структуры алгоритма.

При построении арифметических операторов предпочитают включать в каждый из них определенное законченное действие, например, вычисление определителя, извлечение корня и т. п. На этом этапе последовательность размещения операторов не рассматривается.

Для осуществления вычислений на ЭВМ существенна очередность арифметических операторов, так как один из операторов может обеспечить возможность выполнения следующего. При построении описания вычислительной схемы алгоритма необходимо разместить арифметические операторы в такой последовательности, чтобы выполнение вычислений одного из них обеспечивало возможность реализации следующих за ним операторов. Естественно, что в частных случаях перестановка некоторых операторов местами допускается.

Процесс разработки алгоритма. Каждый разработчик алгоритма, руководствуясь накопленным опытом и знанием закономерностей алгоритмизуемого процесса, изображает его в соответствии со своими представлениями. Однако можно выделить некоторые типичные шаги этой работы и наиболее употребительные средства описания алгоритмов.

■

1. Разработка алгоритма начинается с изучения задания, данного для алгоритмизации. Оно часто представляется в описательной форме с использованием формул, таблиц, графиков и т. п. Перед разработчиками алгоритма возникает необходимость глубоко изучить алгоритмизуемый процесс, уяснить закономерности составляющих его явлений, установить все факторы, влияющие на его течение, оценить степень влияния отдельных явлений на окончательный результат и т. д.

Необходимо определить входную и выходную информацию, задать области изменения аргументов, точность вычислений.

2. Входная информация должна быть полной для обеспечения получения всей выходной информации. Входная информация бывает двух видов: постоянная и переменная. Постоянная входная информация сохраняет значение в процессе счета по всему алгоритму. Переменная информация зависит от конкретной частной задачи, решаемой в данный момент.

При разработке алгоритмов универсальность построенного алгоритма зависит от соотношения объемов переменной и постоянной входной информации. При уменьшении переменной информации, как правило, алгоритм становится менее универсальным.

3. Далее выполняется математическая формализация словесно-описательного условия задачи. Цель ее – построить массивы арифметических {А} и логических {Р} операторов. В массив (Р) входят все условия, которые отражают закономерности алгоритмизуемого процесса. Массивы {А} и {Р} являются тем материалом, из которого строится вычислительная схема алгоритма.

4. Совокупность данных, которые в соответствии с условиями задачи должны быть получены в результате ее решения, составляют выходную информацию.

**Способы** описания алгоритмов. В процессе разработки алгоритмов для их анализа, сокращения числа элементов, удобства программирования и других целей используется много способов описания алгоритмов. Среди них наибольшее распространение получили:

словесный (на естественном языке);

схемы алгоритмов (графический);

операторные схемы; псевдокоды;

языки программирования;

таблицы.

Названные средства описания имеют определенные достоинства и недостатки. Одни из них более удобны на этапе разработки алгоритма, другие – на этапе анализа, третьи – на этапе минимизации алгоритма, четвертые – на этапе общения человека с ЭВМ и т. д.

Дня иллюстрации средств записи алгоритмов приведем простейший пример. Пусть задана последовательность X1, Х2, ..., Xi, *...*, Хn из п произвольных действительных чисел и требуется подсчитать в этой последовательности количество чисел *М* для случая *Xi* ≥ *а* и количество чисел Вдля случая *Xi* < а.

Очевидно, что i может принимать только целые значения 1, 2, ..., п. Отдельные операторы алгоритма будем обозначать числами 1, 2, 3, ... . Они будут играть роль меток (номеров) операторов. Тогда решение задачи можно описать в виде следующего алгоритма:

* 1. *B*:= 0, *M* := 0
  2. *i*:= 1
  3. если *Xi* < *a*, то ПКУ4, иначе ПКУ6
  4. *В* := *В* + 1
  5. ПКУ7
  6. *М* := *М*+1
  7. Если *i* < *n* то ПКУ8, иначе ПКУ10
  8. *i* := *i*+1
  9. ПКУ3
  10. КОНЕЦ

Знак «:=» понимается как символ операции присваивания величине, стоящей слева от него, значения величины, стоящей справа.

Эта последовательность операций и представляет собой запись алгоритма для человека, знающего и правила выполнения арифметических действий. Однако в ней присутствуют и необычные операторы 4, 6 и 8, которые бессмысленны для немашинной математики, поскольку в ней такие равенства, как В = В + 1 и другие, аналогичные ему, невозможны. Суть записи в том, что старое значение Bувеличивается на единицу и новое значение присваивается снова величине с именем *В*. Это равносильно операции, когда результат заносится в заданную клетку таблицы, а вычисления проводятся на черновике. После вычисления нового результата на черновике старый результат в этой клетке таблицы стирается и на его месте записывается новый и т. д. Такая запись порождена удобством для отображения процессов вычислений на машине, где роль таблицы выполняет память машины. В этом случае после вычислений машина направляет новый результат на заранее выбранный адрес памяти, что приводит к автоматическому стиранию старого результата и запоминанию нового.

Воспользуемся данным алгоритмом для иллюстрации различных общепринятых средств его записи. В принятой нами терминологии операторы 1, 2, 4, 6, 8 отнесем к арифметическим (группа А), а 3, 5, 7, 9 – к логическим (группа Р). При необходимости подчеркнуть принадлежность оператора к одной из этих групп будем записывать А1, A2, А4, A6, A8 и соответственно Р3, P5, P7, Р9

Мы остановимся более детально на операторной форме представления алгоритмов и их схем, так как предварительная подготовка задачи для программирования крайне редко обходится без укрупненной разбивки на операторы и описания связей между ними на языке схем. По мере усложнения задач и характера вычислительного процесса, увеличения количества операций, необходимых для их решения, трудности алгоритмизации и программирования резко возрастают. Естественно, что возник вопрос о расчленении вычислительного процесса на такие простые части, чтобы каждую из них программировать отдельно, а затем, соединив частные программы, получить всю программу в целом.

Сложность операторов определяется характером решаемой задачи. Чтобы после объединения операторов получилось описание алгоритмического процесса, необходимо каким-то образом описать взаимосвязь операторов. Обозначим все операторы буквами с индексами.

Для удобства записи логических схем алгоритмов операторы обычно располагают в одну строку и руководствуются следующими правилами:

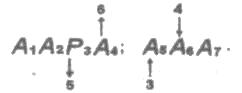
* 1. Порядковый номер оператора в данной схеме изображается нижним индексом оператора. Нумерация операторов сквозная.
  2. Если оператор зависит от параметра, то этот параметр изображается верхним индексом при его буквенном обозначении.
  3. Если знаки двух операторов располагаются в схеме рядом, то оператор, стоящий в схеме слева, передает управление оператору, записанному справа.
  4. Если между двумя записанными рядом знаками операторов стоит точка с запятой, то от оператора, записанного слева, нет передачи управления оператору, записанному справа.
  5. Передача управления оператору, записанному не рядом справа, обозначается стрелкой.

Например, операторная схема может выглядеть так:



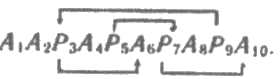
Для удобства записи горизонтальную часть стрелки можно опускать. При этом начало стрелки отмечается малой вертикальной стрелкой, направленной от оператора и снабженной номером оператора, которому передается управление с указанием номера, передающего управление оператора.

Тогда эту же схему можно представить так:



При начертании схем программ обычно принято стрелки от логических операторов изображать над строкой, если логическое условие истинно, и под строкой в противном случае.

В соответствии с указанными правилами приведенный в этом параграфе алгоритм в операторной форме можно записать так:



Во многих случаях бывает удобно группу элементарных операторов обозначать одной буквой. В этом случае придерживаются правила, что управление извне (от операторов, не принадлежащих данной группе) может получать лишь один элементарный оператор группы. Такую гpyппy элементарных операторов называют обобщенным оператором.

С помощью операторной схемы алгоритма составляется схема решения задачи, которая дополняется операторами, свойственными машинному вычислительному процессу. Для каждого оператора в порядке его записи затем составляется своя частная программа. Совокупность этих частных программ и дает программу решения задачи. Использование операторной схемы алгоритма при программировании рассчитано на получение следующих преимуществ:

1) облегчается работа по составлению программы;

2) уменьшается количество ошибок при программировании;

3) наличие операторной схемы алгоритма облегчает проверку готовых программ;

4) появляется возможность возобновлять работу над программой после длительного перерыва или поручать продолжение начатой работы другому лицу.

Среди всех задач выделяют расчетные, алгоритмы которых легко описываются с помощью математических формул, и логические задачи.

К логическим задачам относятся преимущественно многочисленные задачи управления и планирования. При их решении на ЭВМ схемы алгоритмов приобретают важное значение так как они являются пока почти единственным формальным аппаратом, отображающим динамику сложных процессов.

**Схема алгоритма**, с одной стороны, служит как формальный язык для записи содержания и логических связей задачи, а с другой – она сравнительно легко преобразуется в программу дня ЭВМ.

Схемы алгоритмов – функционально ориентированные графические изображения, с помощью которых, используя текст и специальные символы, описывают последовательность шагов процесса во времени, связи рассматриваемой системы с внешней средой и ветвление процесса. Краткое текстовое или формальное описание шагов приводится внутри символов. Сведения о входных и выходных данных некоторых шагов процесса могут даваться в закодированной форме. Схемы алгоритмов обладают тем преимуществом, что для их понимания требуется специальных знаний.

Дна удобства чтения схем алгоритмов в ряде стран разработаны стандартные графические обозначения для наиболее часто употребляемых и специфических операторов.

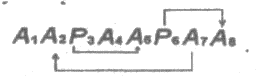
Поясним понятие схемы алгоритма. Пусть требуется вычислить функцию

для всех дискретных значений *х*, с шагом 0,1 из сегмента [-7; 5].

Так как у представлен через аналитические выражения и , по которым могут быть произведены вычисления в результате подстановки в них значений *хi* то алгоритм для вычисления очевиден, остается построить схему алгоритма. Из условий задачи вытекает необходимость выполнения следующих операций.

* + 1. Ввести исходное значение *х* := *Х*0:= –7. ПКУ2.
    2. *М* := *х*2 ПКУЗ.
    3. Проверить . Если да, то ПКУ4; если нет, то ПКУ5.
    4. . ПКУ6.
    5. . ПКУ6.
    6. Проверить х = 5. Если да, то ПКУ8; если нет, то ПКУ7.
    7. . ПКУ2.
    8. Конец.

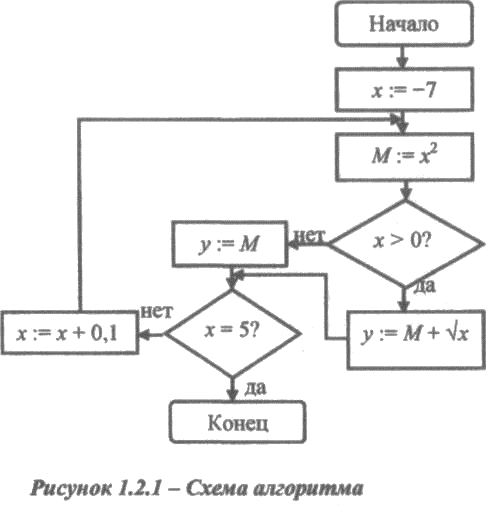
Обозначим эти восемь операций соответственно А1, А2,*P*3, A4, A5, P6, A7, A8 (или еще короче: 1, 2, ..., 8). Если расположить эти символы в строчку и с помощью стрелок указать связи, то получим операторную схему алгоритма:



Если теперь вместо символов операторов вычертить прямоугольники, в них написать содержание действий, выполняемых данными операторами, и показать связями последовательность этих действий, то получим схему алгоритма (рис. 1.2.4).

**Граф-схема алгоритма** (ГСА) – конечный связный ориентированный граф G = (V, Е), где V – множество вершин графа *vi* ∈ V, 1 ≤ i ≤ N, которые соответствуют операторам; Е – множество дуг графа , , которые задают порядок следования операторов; N = | V | – число вершин графа; *М* = | Е | – число дуг графа.

Таким образом, ГСА есть графическое изображение последовательности операций, согласно которым получают решение задачи. Каждый этап процесса обработки информации представляется в виде геометрических символов (блоков), имеющих определенную конфигурацию в зависимости от характера выполняемых операций. Перечень символов, их наименование, отображаемые ими функции, форма и размеры определяются ГОСТами.



Основные свойства граф-схем алгоритмов следующие:

* + - 1. Граф-схема состоит из конечного числа точек, называемых вершинами (узлами), и соединяющих их стрелок.
      2. В граф-схеме имеются два особых узла: входной, в который не входит ни одна стрелка, и выходной, из которого не выходит ни одна стрелка.
      3. Все узлы граф-схем отмечены своей логической переменной *Рi*, или арифметическим оператором Аj.
      4. Из каждого узла *Рi* отходят, как правило, две стрелки, а с узла Аj – только одна стрелка.

Если в ГСА логические переменные и арифметические операторы заменить соответствующими описаниями, то граф-схема превратится в схему алгоритма (рис. 1.1.5).

*A*1

*A*2

*P*3

*A*4

*P*6

*A*5

*A*7

*A*8

Рисунок 1.2.2 – Граф-схема алгоритма

При всем многообразии алгоритмов решения задач в них можно выделить три основных вида вычислительных процессов: линейный; ветвящийся; циклический. Линейным называется вычислительный процесс, при котором все этапы решения задачи выполняются в естественном порядке следования записи этих этапов. Ветвящимся называется такой вычислительный процесс, в котором выбор направления обработки информации зависит от исходных или промежуточных данных (от результатов проверки выполнения какого-либо логического условия). Циклом называется многократно повторяемый участок вычислений.

**Модель представления параллельного алгоритма.** Параллельная обработка информации выполняется на кластере, в общем случае гетерогенном. Он представляется множеством *P* = {*p*1, *p*2, …, *pm*} где *pi*, – отдельный узел (компьютер) кластера, *т* – число узлов кластера. Каждый компьютер рi имеет характеристику производительности si, которая определяет время выполнения одной условной единицы вычислений. Узлы кластера соединены коммуникационными каналами. Каждый канал между узлами *pi* и *pj*, обозначенный *lij* имеет характеристику, которая определяет ширину канала и скорость передачи данных между *pi* и *pj*.

Поток, обрабатываемый параллельным алгоритмом, представляется множеством объектов *J* = {*j*1, …, *jn*}. Назовем алгоритм обработки объекта определенного типа сценарием. Сценарии обработки, в общем случае, содержат множество операций, каждая из которых может избирательно применяться к объектам различных типов, либо применяться с различными параметрами в зависимости от типа обрабатываемого объекта. Система обработки способна выполнять множество операций обработки О = {о1, ...,*ok*}, k > т. Сценарий обработки отдельного типа кадров содержит подмножество операций О*j* = {*оj*1, ..., *оjk*}, . Операции в каждом сценарии упорядочены отношениями следования т.е. для кадра с типом j операция *a* выполняется перед операцией *b*. Каждая операция обработки *оi* характеризуется сложностью вычислений , представленной количеством единиц вычисления данной операций для определенного типа кадров j. Две операции над различными кадрами, назначенными на один и тот же процессор, не могут выполняться одновременно. Два экземпляра одной и той же операции над различными кадрами также не могут выполняться в один и тот же промежуток времени.

Все сценарии обработки представляются в форме направленного ациклического графа в виде кортежей G = (V, E, W, C), где

*V* – множество вершин графа . Каждая вершина ассоциируется с операцией обработки данных. Множество вершин представляет декомпозицию параллельной программы обработки потока изображений на отдельные операции (гранулы параллелизма);

Е –множество дуг графа {*еij* = (*vi*, *vj*)} ∈ *E*, *i*, *j* = 1, …, *N*, *i* ≠ *j*. Дуга представляет собой отношение предшествования между операциями алгоритма и определяет передачу результатов обработки от источника к приемнику;

*W* – матрица вычислительной стоимости операций ;

*C* – множество стоимости дуг, элемент *cij* ∈ *C* определяет объем информации между двумя операциями обработки данных, передаваемой по дуге *еij* ∈ *E*. Связанные дугой операции используют один формат данных, поэтому для всех сценариев и типов кадров дуги имеют равную стоимость.

Построение приложения для параллельной обработки потока данных включает: создание сценариев, описывающих логическую структуру программы; назначение операций обработки вершинам графа и определение параметров вызова для каждого типа данных; отображение графа сценариев на топологию вычислительного кластера. Параллельная программа предсгавпяетм преобразованием , где . Каждое подмножество операций размещается на выбранном узле кластера.

Теперь цель обработки потока можно записать как минимизацию времени обработки N объектов данных:

Тopt = min{max TN}, (1.2.1)

где TN означает момент окончания обработки объекта N. Оптимизационный алгоритм использует для оценки расписания имитационную модель, эквивалентную реальному вычислительному процессу.

Вычислительные гранулы объединяются в специализированные библиотеки, которые динамически подключаются при выполнении параллельного приложения. Реализация различных классов алгоритмов обработки информации в виде библиотек вычислительных гранул позволяет расширить сферу применения системы организации параллельных вычислений и дает возможность создавать новые классы приложений.

Этапы параллельного проектирования. Получить параллельный алгоритм решения задачи можно путем распараллеливания имеющегося последовательного алгоритма или путем разработки нового параллельного алгоритма. Возможно, для осуществления распараллеливания алгоритм решения задачи придется заменить или модифицировать (например, устранить некоторые зависимости между операциями).

Разработка алгоритмов параллельных вычислений состоит в следующем:

* Выполнить анализ имеющихся вычислительных схем и осуществить их разделение (декомпозицию) на части (подзадачи), которые могут быть реализованы в значительной степени независимо друг от друга.
* Выделить для сформированного набора подзадач информационные взаимодействия, которые должны осуществляться в ходе решения исходной поставленной задачи.
* Определить доступную для решения задачи вычислительную систему и выполнить распределение имеющего набора подзадач между процессорами системы.
* Разделить вычисление на независимые части.
* Выделить информационное взаимодействие между частями.
* Масштабировать задачи.
* Распределить все задачи между процессорами.

При рассмотрении параллельных алгоритмов вводится понятие **параллельной ГСА,** в состав которой входят вершины распараллеливания/синхронизации, функциональность которых обычно совмещается (фрагмент такой ГСА приведен на рис. 1.1.6). Иногда в состав ГСА вводятся вершины дополнительных типов с целью обеспечения возможности моделирования выполнения алгоритма сетью Петри. Однако не любой ориентированный граф, составленный из вершин, указанных выше типов, может быть отождествлен с корректным алгоритмом. Например, из операторной вершины не может выходить более одной дуги. Поэтому на практике обычно ограничиваются рассмотрением подкласса граф-схем алгоритмов, удовлетворяющих свойствам безопасности, живости и устойчивости.

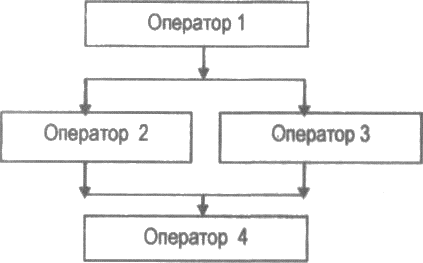


Рисунок 1.2.3 – Фрагмент параллельной ГСА