**Алгоритмы и сложность**

Алгоритм — это конкретная процедура для решения четко определенной вычислительной задачи. Разработка и анализ алгоритмов лежат в основе всех аспектов информатики: искусственного интеллекта, баз данных, графики, сетей, операционных систем, безопасности и так далее. Разработка алгоритмов — это больше, чем просто программирование. Она требует понимания доступных альтернатив для решения вычислительной задачи, включая аппаратное обеспечение, сеть, язык программирования и ограничения производительности, которые сопровождают любое конкретное решение. Она также требует понимания того, что значит для алгоритма быть «правильным» в том смысле, что он полностью и эффективно решает поставленную задачу.

Сопутствующее понятие — это разработка конкретной структуры данных, которая позволяет алгоритму работать эффективно. Важность структур данных связана с тем фактом, что основная память компьютера (где хранятся данные) является линейной и состоит из последовательности ячеек памяти, последовательно пронумерованных 0, 1, 2,…. Таким образом, простейшая структура данных представляет собой линейный массив, в котором соседние элементы пронумерованы последовательными целочисленными «индексами», а значение элемента доступно по его уникальному индексу. Массив можно использовать, например, для хранения списка имен, а эффективные методы необходимы чтобы эффективно искать и извлекать определенное имя из массива. Например, сортировка списка в алфавитном порядке позволяет использовать так называемый метод бинарного поиска, при котором оставшаяся часть списка для поиска на каждом шаге разрезается пополам. Этот метод поиска похож на поиск определенного имени в телефонной книге. Зная, что книга находится в алфавитном порядке, можно быстро перейти на страницу, близкую к странице, содержащей нужное имя. Многие алгоритмы были разработаны для эффективной сортировки и поиска списков данных.

Хотя элементы данных хранятся в памяти последовательно, они могут быть связаны друг с другом указателями (по сути, адресами памяти, хранящимися вместе с элементом, чтобы указать, где находится следующий элемент или элементы в структуре), так что данные могут быть организованы таким же образом, как и при доступе к ним. Простейшая такая структура называется связанным списком, в котором к не смежно сохраненным элементам можно получить доступ в заранее заданном порядке, следуя указателям от одного элемента в списке к следующему. Список может быть циклическим, когда последний элемент указывает на первый, или каждый элемент может иметь указатели в обоих направлениях, образуя двусвязный список. Были разработаны алгоритмы для эффективного управления такими списками путем поиска, вставки и удаления элементов.

Указатели также предоставляют возможность реализации более сложных структур данных. Например, граф представляет собой набор узлов (элементов) и связей (известных как ребра), соединяющих пары элементов. Такой граф может представлять собой набор городов и соединяющих их автомагистралей, расположение элементов схемы и соединительных проводов на чипе памяти или конфигурацию людей, взаимодействующих через социальную сеть. Типичные алгоритмы графов включают в себя стратегии обхода графа, например, как следовать по ссылкам от узла к узлу (возможно, в поисках узла с определенным свойством) таким образом, чтобы каждый узел посещался только один раз. Связанная с этим проблема — определение кратчайшего пути между двумя заданными узлами произвольного графа. (Смотрите теорию графов.) Например, проблема, представляющая практический интерес в сетевых алгоритмах, состоит в том, чтобы определить, сколько «сломанных» связей можно допустить до того, как связь начнет прерываться. Аналогичным образом, при проектировании микросхем сверхбольшой интеграции (СБИС) важно знать, является ли граф, представляющий схему, плоским, то есть можно ли его нарисовать в двух измерениях без пересечения каких-либо связей (соприкосновения проводов).

(Вычислительная) сложность алгоритма — это мера количества вычислительных ресурсов (времени и пространства), которые конкретный алгоритм потребляет при работе. Ученые используют математические меры сложности, которые позволяют им предсказать до написания кода, насколько быстро будет работать алгоритм и сколько памяти ему потребуется. Такие прогнозы являются важным руководством для программистов, реализующих и выбирающих алгоритмы для реальных приложений. Вычислительная сложность — это континуум, поскольку для некоторых алгоритмов требуется линейное время (то есть требуемое время увеличивается непосредственно с количеством обрабатываемых элементов или узлов в списке, графе или сети), тогда как другим требуется квадратичное или даже экспоненциальное время для завершения (то есть требуемое время увеличивается с увеличением количества элементов в квадрате или с экспоненциальным увеличением этого числа). На дальнем конце этого континуума лежат мутные моря трудноразрешимых проблем – тех, решения которых не могут быть эффективно реализованы. Для решения этих задач ученые стремятся найти эвристические алгоритмы, которые почти могут решить проблему и выполниться за приемлемое время.

Еще дальше находятся те алгоритмические проблемы, которые можно сформулировать, но не решить; то есть можно доказать, что для решения задачи нельзя написать программу. Классическим примером неразрешимой алгоритмической проблемы является проблема остановки, которая утверждает, что не может быть написана программа, которая могла бы предсказать, остановится ли какая-либо другая программа после конечного числа шагов. Неразрешимость проблемы остановки имеет непосредственное практическое значение для разработки программного обеспечения. Например, было бы легкомысленно пытаться разработать программный инструмент, который предсказывает, есть ли в другой разрабатываемой программе бесконечный цикл (хотя наличие такого инструмента было бы чрезвычайно полезно).

**Архитектура и организация**

Компьютерная архитектура имеет дело с проектированием компьютеров, устройств хранения данных и сетевых компонентов, которые хранят и выполняют программы, передают данные и управляют взаимодействием между компьютерами, сетями и с пользователями. Архитекторы используют параллелизм и различные стратегии организации памяти для проектирования вычислительных систем с очень высокой производительностью. Компьютерная архитектура требует сильного взаимодействия между учеными и инженерами, поскольку и те, и другие фундаментально сосредоточены на проектировании аппаратного обеспечения.

На самом фундаментальном уровне компьютер состоит из устройства управления, арифметико-логического устройства (АЛУ), блока памяти и контроллеров ввода/вывода. АЛУ выполняет простые операции сложения, вычитания, умножения, деления и логические операции, такие как ИЛИ и И. В памяти хранятся инструкции и данные программы. Устройство управления извлекает данные и инструкции из памяти и использует операции АЛУ для выполнения этих инструкций с использованием этих данных. (Устройство управления и АЛУ вместе называются центральным процессором [ЦП].) Когда встречается команда ввода или вывода, устройство управления передает данные между памятью и назначенным контроллером ввода-вывода. Скорость работы процессора в первую очередь определяет скорость компьютера в целом. Все эти компоненты — устройство управления, АЛУ, память и контроллеры ввода-вывода — реализованы на транзисторных схемах.

Компьютеры также имеют другой уровень памяти, называемый кэшем, небольшой, чрезвычайно быстрый (по сравнению с основной памятью или оперативной памятью [ОЗУ]) блок, который можно использовать для хранения информации, которая требуется срочно или часто. Текущие исследования включают разработку кэша и алгоритмы, которые могут предсказать, какие данные могут потребоваться в следующий раз, и предварительно загрузить их в кеш для повышения производительности.

Контроллеры ввода-вывода подключают компьютер к конкретным устройствам ввода (таким как клавиатуры и сенсорные дисплеи) для передачи информации в память, а также устройствам вывода (таким как принтеры и дисплеи) для передачи информации из памяти пользователям. Дополнительные контроллеры ввода-вывода подключают компьютер к сети через порты, которые обеспечивают канал передачи данных, когда компьютер подключен к Интернету.

К контроллерам ввода-вывода подключены дополнительные устройства хранения данных, такие как дисковод, которые медленнее и имеют большую емкость, чем основная или кэш-память. Дисководы используются для хранения постоянных данных. Они могут быть либо постоянно, либо временно подключены к компьютеру как CD-диск, DVD-диск или карты памяти (также называемой флэш-накопителем).

Работа компьютера после загрузки программы и некоторых данных в оперативную память происходит следующим образом. Первая команда передается из оперативной памяти в устройство управления и интерпретируется аппаратной схемой. Например, предположим, что инструкция представляет собой строку битов, которая является кодом LOAD 10. Эта инструкция загружает содержимое ячейки памяти 10 в АЛУ. Выбирается следующая инструкция, скажем, ADD 15. Устройство управления загружает содержимое ячейки памяти 15 в АЛУ и добавляет его к уже имеющемуся там числу. Наконец, инструкция STORE 20 сохранит эту сумму в ячейке 20. На этом уровне работа компьютера мало чем отличается от работы карманного калькулятора.

В общем, программы — это не просто длинные последовательности команд LOAD, STORE и арифметических операций. Самое главное, что компьютерные языки включают условные инструкции — по сути, правила, которые гласят: «Если ячейка памяти *n* удовлетворяет условию *а*, делай инструкцию номер *x* следующей, в противном случае делай инструкцию *y*». Это позволяет определять ход программы по результатам предыдущих операций — критически важная способность.

Наконец, программы обычно содержат последовательности инструкций, которые повторяются несколько раз, пока предопределенное условие не станет истинным. Такая последовательность называется циклом. Например, цикл требуется для вычисления суммы первых *n* целых чисел, где *n* — значение, хранящееся в отдельной ячейке памяти. Компьютерные архитектуры, которые могут выполнять последовательности инструкций, условные инструкции и циклы, называются «полными по Тьюрингу», что означает, что они могут выполнять любой алгоритм, который можно определить. Полнота по Тьюрингу является фундаментальной и важнейшей характеристикой любой организации компьютера.

Логическое проектирование — это область информатики, которая занимается проектированием электронных схем с использованием фундаментальных принципов и свойств логики (смотрите Булева алгебра) для выполнения операций устройства управления, АЛУ, контроллеров ввода-вывода и другого аппаратного обеспечения. Каждая логическая функция (И, ИЛИ и НЕ) реализуется устройством определенного типа, называемым вентилем. Например, схема сложения АЛУ имеет входы, соответствующие всем битам двух чисел, которые необходимо сложить, и выходы, соответствующие битам суммы. Расположение проводов и вентилей, соединяющих входы с выходами, определяется математическим определением сложения. В конструкции устройства управления предусмотрены схемы, интерпретирующие инструкции. Ввиду необходимости повышения эффективности при проектировании логики также необходимо оптимизировать схему, чтобы она функционировала с максимальной скоростью и имела минимальное количество вентилей и схем.

Важной областью, связанной с архитектурой, является проектирование микропроцессоров, которые представляют собой полноценные центральные процессоры — устройство управления, АЛУ и память — на одной интегральной микросхеме. Дополнительная память и схемы контроля ввода-вывода связаны с этим чипом, чтобы сформировать полноценный компьютер. Эти устройства размером с ноготь содержат миллионы транзисторов, которые реализуют вычислительные блоки и модули памяти современных компьютеров.

Проектирование микропроцессора СБИС происходит в несколько стадий, которые включают в себя создание исходной функциональной или поведенческой спецификации, кодирование этой спецификации на языке описания аппаратного обеспечения, разбиение проекта на модули и создание размеров и форм для возможных компонентов микросхемы. Это также включает в себя планирование микросхемы, которое включает в себя построение «плана этажа», чтобы указать, где на кристалле следует разместить каждый компонент и соединить его с другими компонентами. Ученые также участвуют в создании систем автоматизированного проектирования (САПР, CAD), которые помогают инженерам на различных этапах проектирования микросхем и в разработке необходимых теоретических результатов, например, как эффективно спроектировать «план этажа» с практически минимальной площадью, удовлетворяющей требованиям заданных ограничений.

Достижения в области технологии интегральных схем были невероятными. Например, в 1971 году первый микропроцессорный чип (4004 корпорации Intel) имел всего 2300 транзисторов, в 1993 году чип Intel Pentium имел более 3 миллионов транзисторов, а к 2000 году количество транзисторов в таком чипе составляло около 50 миллионов. Чип Power7, представленный IBM в 2010 году, содержал около 1 миллиарда транзисторов. Феномен удвоения количества транзисторов в интегральной схеме примерно каждые два года, широко известно как закон Мура.

Отказоустойчивость — это способность компьютера продолжать работу при выходе из строя одного или нескольких его компонентов. Чтобы обеспечить отказоустойчивость, ключевые компоненты часто дублируются, чтобы резервный компонент мог взять на себя работу при необходимости. Такие приложения, как управление воздушным судном и управление производственными процессами, работают на системах с резервными процессорами, готовыми взять на себя управление в случае отказа основного процессора, а резервные системы часто работают параллельно, поэтому переход происходит плавно. Если системы являются критически важными, поскольку их отказ может быть потенциально катастрофическим (как в случае управления воздушным судном), несовместимые результаты, полученные от дублирующих процессов, работающих параллельно на отдельных машинах, разрешаются с помощью механизма голосования. Ученые участвуют в анализе таких систем резервирования, предоставляя теоретические подходы к оценке надежности, достигаемой с помощью заданной конфигурации и параметров процессора, таких как среднее время между сбоями и среднее время, необходимое для ремонта процессора. Отказоустойчивость также является желательной функцией в распределенных системах и сетях. Например, преимущество распределенной базы данных состоит в том, что данные, зарезервированные на разных узлах сети, могут обеспечить естественный механизм резервного копирования в случае сбоя одного узла.