Практика 3-4 Анализ алгоритмов (продолжение)

Теоретическая часть.

Рекурсивные функции

Когда появились задачи, для которых не были найдены методы их решения, то возник вопрос: отсутствие алгоритма решения для данного класса задач - это результат недостаточности знаний или решающего алгоритма для этого класса задач не существует?

Именно тогда для решения этой проблемы ввели понятие вычислимой функции.

Пусть функция y зависит от целочисленных аргументов $x_1, x_2, ..., x_n$.

Функция y = f($x_1, x_2, ..., x_n$) называется эффективно вычислимой, если существует алгоритм, который позволяет вычислить ее значение.

Простейшие эффективно вычислимые функции:

1. нуль-функция (оператор аннулирования) O(x)=0

2. функция следования (оператор сдвига) $\lambda(x) = x+1$

3.функция выбора аргумента (оператор проектирования)

$$I_n^m(x_1, x_2, ..., x_n) = x_m, 1 <= m <= n$$

Функции делятся на примитивно рекурсивные, частично рекурсивные и общерекурсивные.

Функция f называется примитивно рекурсивной, если ее можно получить конечным числом операций подстановки и примитивной рекурсии, используя лишь простейшие функции 1,2,3. Операции подстановки(суперпозиции) и примитивной рекурсии, если их применить ко всюду определенным функциям, дадут в результате снова всюду определенные функции. Это означает, что все примитивно рекурсивные функции всюду определены.

Функция f называется частично рекурсивной .если она может быть получена за конечное число шагов из простейших функций 1,2,3 при помощи операции суперпозиции , схем примитивной рекурсии и операции минимизации. Это значит, что частично рекурсивная функция определяется аналогично примитивно рекурсивной, только к двум операторам суперпозиции и примитивной рекурсии добавляется ещё третий оператор — минимизации аргумента.

Функция f называется общерекурсивной, если она частично рекурсивна и всюду определена

Класс частично рекурсивных функций шире класса примитивно рекурсивных функций, так как все примитивно рекурсивные функции всюду определены, чего нельзя сказать о частично рекурсивных функциях.

Частично рекурсивные функции – одно из главных понятий теории алгоритмов.

Каждая частично рекурсивная функция вычислима путем определенной процедуры механического характера (то есть именно то, что мы интуитивно ждем от алгоритма).

Кроме того, оказалось, что числовые функции, вычисленные до сих пор с помощью любых классов алгоритмов, были частично рекурсивными.

Поэтому общепринята гипотеза – тезис Черча:

Каждая интуитивно вычислимая функция является частично рекурсивной.

Эту гипотезу нельзя доказать, но можно опровергнуть, построив хотя бы один контрпример. Пока таких примеров построено не было.

Замечние:

В терминах императивного программирования — *примитивно рекурсивные функции* соответствуют программным блокам, в которых используется только арифметические операции, а также условный оператор и оператор арифметического цикла (оператор цикла, в котором число итераций известно на момент начала цикла).

Если же программист начинает использовать оператор цикла while, в котором число итераций заранее неизвестно и, в принципе, может быть бесконечным, то он переходит в класс *частично рекурсивных функций.*

. С точки зрения императивного программирования, результатом частично рекурсивной функции может быть не только число, но и исключение или уход в бесконечный цикл, соответствующие неопределённому значению.

Дополнительные замечания.

Теория вычислимости, также известная как теория рекурсивных функций, — это раздел современной математики, лежащий на стыке математической логики, теории алгоритмов и информатики, возникший в результате изучения понятий вычислимости и невычислимости. Изначально теория была посвящена вычислимым и невычислимым функциям и сравнению различных моделей вычислений. Сейчас поле исследования теории вычислимости расширилось — появляются новые определения понятия вычислимости и идёт слияние с математической логикой, где вместо вычислимости и невычислимости идёт речь о доказуемости и недоказуемости (выводимости и невыводимости) утверждений в рамках каких-либо теорий.

Теория вычислимости берёт свое начало от диссертации Тьюринга (1936), в которой он ввел понятие абстрактной вычислительной машины, получившей впоследствии его имя, и

доказал фундаментальную теорему о неразрешимости задачи о её остановке. Знаменитая теорема Гёделя о неполноте (1931) была доказана в терминах примитивно рекурсивных функций, класс которых в 1934 году Гёдель расширил до класса общерекурсивных функций. Формализм, развитый Гёделем оказался эквивалентным тьюринговскому (а также многим другим). Вместе с Тезисом Чёрча — Тьюринга этот факт явно продемонстрировал содержательность новой теории, и сейчас эти определения общеприняты в качестве формального аналога алгоритмически вычислимых функций.

Рекурсивные алгоритмы

1. Логарифмические тождества

При анализе рекурсивных алгоритмов достаточно часто используются логарифмические тождества, далее предполагается, что, a > 0, b > 0, c > 0, основание логарифма не равно единице:

$$b^{\log_b a} = a; \quad e^{\ln x} = x; \quad \log_b a^c = c * \log_b a; \quad \log_b a = 1/\log_a b$$

$$\log_b a = \log_c a / \log_c b \Rightarrow$$

в записи Q(ln(x)) основание логарифма не существенно, если он больше единицы, т.к. константа скрывается обозначением Q.

$$a^{\log_b c} = c^{\log_b a}$$

2. Рекурсивные алгоритмы.

Основной метод построения рекурсивных алгоритмов – это метод декомпозиции. Идея метода состоит в разделении задачи на части меньшей размерности, получение решение для полученных частей и объединение решений.

В общем виде, если происходит разделение задачи на b подзадач, которое приводит к необходимости решения a подзадач размерностью n/b, то общий вид функции трудоемкости имеет вид:

fA(n)= a * fA(n/b)+d(n)+U(n) (*), где:

d(n) – трудоемкость алгоритма деления задачи на подзадачи, U(n) – трудоемкость алгоритма объединения полученных решений.

Рассмотрим, например, известный алгоритм сортировки слиянием, принадлежащий Дж. Фон Нейману:

На каждом рекурсивном вызове переданный массив делится пополам, что дает оценку для d(n) =Q (1), далее рекурсивно вызываем сортировку полученных массивов половинной длины (до тех пор, пока длина массива не станет равной единице), и сливаем возвращенные отсортированные массивы за Q (n).

Тогда ожидаемая трудоемкость на сортировку составит: fA(n)= 2 * fA(n/2)+ Q (1)+ Q (n)

Тем самым возникает задача о получении оценки сложности функции трудоемкости, заданной в виде (*), для произвольных значений a и b.

Асимптотический анализ функций

При анализе поведения функции трудоемкости алгоритма часто используют принятые в математике асимптотические обозначения, которые позволяют показать скорость роста функции, не показывая конкретные коэффициенты.

Такую оценку называют сложностью алгоритма. Эта оценка позволяет выбрать тот или иной алгоритм для больших значений размерности исходных данных. (смотри практику 1-2).

Добавьте упражнение к практике 1-2.

Возьмите два алгоритма: сортировку вставками и сортировку слиянием. Проанализируйте эти алгоритмы по асимптотике роста..