Понятие о сложности вычислений

Выполнила: студентка группы 4252 Савельева Н.К.

Под алгоритмом обычно понимают четко определенную последовательность действий, приводящую через конечное число шагов к результату — решению задачи, для которой разработан алгоритм.

Основные свойства, присущие любому алгоритму:

1. массовость — алгоритм предназначен для решения задачи с некоторым множеством допустимых входных данных;

2. конечность — алгоритм должен завершаться за конечное число шагов (но это количество шагов может быть разным для разных входных данных).

Задачи могут быть сформулированы по-разному (дифференциальные уравнения, задачи на графах, задачи оптимизации и т.п.).

Для того чтобы можно было строить единую теорию алгоритмов, необходимо свести разные формулировки задач к какому-то «единому знаменателю».

Например, можно считать, что задача сводится к вычислению некоторой функции $F:X \to Y$. Ясно, что в таком виде можно сформулировать любую задачу. Но для некоторых задач функция F может быть выражена неявно. Например, для задачи поиска минимума функции ϕ на отрезке [0,1] имеем: X = множество функций, Y = [0,1], $F(\phi) = x^*: \phi(x^*) = \min\{\phi(x): x \in [0,1]\}$.

Количественная характеристика потребляемых ресурсов, необходимых программе или алгоритму для работы (успешного решения задачи) — это и есть сложность алгоритма.

Основные ресурсы: время (временнбя сложность) и объем памяти (ёмкостная сложность). Наиболее важной (критической) характеристикой является время. Очевидно, что для разных экземпляров задачи (для разных входных данных) алгоритму может требоваться разное количество ресурсов.

С каждым экземпляром x задачи Z связывается определенное число (реже — набор чисел) |x|, называемое длиной или размером входных данных (размером задачи). Размер задачи — это объем входных данных, необходимых для задания всех параметров задачи.

Однако для многих задач количество времени, необходимое для решения задачи, зависит не только от размера входных данных, но и от самих данных. То есть для решения задачи для двух входных данных x и y одинакового размера (|x|=|y|) алгоритм может тратить разное время.

Поэтому для получения оценок временной сложности в зависимости от размера задачи определяют ее как максимальное время, затрачиваемое алгоритмом для входных данных длины n: $T(n)=\max\{T(x): |x|=n\}$. Эта функция называется сложностью алгоритма в худшем случае.

Формально сложность в среднем определяется так:

$T_{ср}(n)=\Sigma T(x)p(x)$

где $p(x)$ — вероятность появления входных данных x, а суммирование ведется по всем возможным входным данным размера n. К сожалению, только для небольшого количества задач (например, для задачи сортировки) удается найти естественный способ определения вероятностей для входных данных. Поэтому при оценке сложности алгоритма обычно рассматривают его сложность в худшем случае.

Более того, обычно оценивают не точное значение функции сложности $T(n)$, а порядок роста этой функции, т. е. находят такую функцию $f(n)$, что $T(n) = O(n)$ при $n \to \infty$.

Классы сложности задач

Теорема Блюма об ускорении:

Существует такая алгоритмически разрешимая задача Z, что любой алгоритм A, решающий задачу Z, можно ускорить следующим образом: существует другой алгоритм A′, также решающий Z и такой, что $T_{A'}(n) \leq \log T_A(n)$ для почти всех n.

Классами сложности называются множества вычислительных задач, примерно одинаковых по сложности вычисления.

Существуют классы сложности языков и функциональные классы сложности. Класс сложности языков — это множество предикатов (функций, получающих на вход слово и возвращающих ответ 0 или 1), использующих для вычисления примерно одинаковые количества ресурсов. Понятие функционального класса сложности аналогично, за исключением того, что это не множество предикатов, а множество функций. В теории сложности, по умолчанию, класс сложности — это класс сложности языков. Типичное определение класса сложности выглядит так:

Классом сложности X называется множество предикатов P(x), вычислимых на машинах Тьюринга и использующих для вычисления O(f(n)) ресурса, где n — длина слова x.

В качестве ресурсов обычно берутся время вычисления (количество рабочих тактов машины Тьюринга) или рабочая зона (количество использованных ячеек на ленте во время работы). Языки, распознаваемые предикатами из некоторого класса (то есть множества слов, на которых предикат возвращает 1), также называются принадлежащими тому же классу.

Для каждого класса существует категория задач, которые являются «самыми сложными». Это означает, что любая задача из класса сводится к такой задаче, и притом сама задача лежит в классе. Такие задачи называют полными задачами для данного класса.

Класс P (задачи с полиномиальной сложностью)

Класс P (от англ. polynomial) — множество задач распознавания, которые могут быть решены на детерминированной машине Тьюринга за полиномиальное от длины входа время.

Иными словами, задача относится к классу P, если существует константа k и алгоритм, решающий задачу с:

$F_a(n)=O(n^k)$,

где n – длина входа алгоритма в битах n = |D| [6].

Отметим следующие преимущества алгоритмов из этого класса:

для большинства задач из класса P константа k меньше 6;

класс P инвариантен по модели вычислений (для широкого класса моделей);

класс P обладает свойством естественной замкнутости (сумма или произведение полиномов есть полином).

Таким образом, задачи класса P есть уточнение определения «практически разрешимой» задачи.

Класс NP (полиномиально проверяемые задачи)

Рассмотрим задачу о сумме: Дано N чисел – A = (a_1,…a_n) и число V. Задача: Найти вектор (массив) X=(x_1,…,x_n), $x_i \in \{0,1\}$, такой, что $\sum a_i x_i = V$.

Содержательно: может ли быть представлено число V в виде суммы каких либо элементов массива A. Если какой-то алгоритм выдает результат – массив X, то проверка правильности этого результата может быть выполнена с полиномиальной сложностью: проверка $\sum a_i x_i = V$ требует не более Θ (N) операций.

Формально: $\forall\ D \in D_A$, |D|=n поставим в соответствие сертификат $S \in S_A$, такой что |S|=O (n^l) и алгоритм $A_s = A_s$ (D,S), такой, что он выдает «1», если решение правильно, и «0», если решение неверно. Тогда задача принадлежит классу NP, если F (A_s)=O (n^m) [6].

Основы теории сложности вычислений

Теория сложности вычислений — бурно развивающаяся область теоретической информатики (theoretical computer science) и охватывает как чисто теоретические вопросы, так и вопросы, непосредственно связанные с практикой.

Среди наиболее важных приложений этой теории можно назвать способы построения и анализа эффективных алгоритмов, а также современные криптографические методы.

Нормальные алгорифмы Маркова

Норма́льный алгори́тм Ма́ркова (НАМ)— один из стандартных способов формального определения понятия алгоритма, так же как и машина Тьюринга. Понятие нормального алгоритма введено А. А. Марковым в конце 1940-х годов.

Задача для алгоритмов Маркова ставится в виде: найти алгоритм (написать программу) переводящую любую строку S (заданную на некотором алфавите (т.е. наборе символов, которые могут в нее входить)) из некоторого допустимого множества входных строк в строку f(S). Т.е., построить программу - преобразователь строк, выполняющую некое преобразование.

Например: перевести все буквы в строке в верхний регистр, инвертировать регистр, перевернуть строку (reverse), и даже такую задачу: из строкового представления десятичного числа получить строковое представление числа на единицу больше (или в 2 раза больше).

Программа на языке алгоритмов Маркова - представляет из себя набор правил (Rules). Каждое правило представляет собой замену. Т.е. правило имеет вид:

S1 -> S2, где S1 и S2 некие строки.

Правило представляет подстановки, последовательно применяемые ко входной строке и приводящие в итоге ее к требуемой выходной строке. Порядок задания правил важен.

Работает алгоритм этот следующим образом. Берется исходная строка и мы начинаем перебирать правила с самого первого, анализируя, может ли оно быть применено (существует ли в строке S подстрока S1). Если не может -> анализируется следующее по порядку правило. Если не одно правило не подошло, алгоритм завершается, текущее состояние строки S является результатом работы алгоритма.

Если же правило применимо - совершается замена самого левого вхождения подстроки S1 на строку S2 (или, выражаясь языком питона, S = S.replace(S1, S2, 1)). Причем далее правила начинают перебираться опять с начала.

Еще есть так называемые терминальные правила, обозначающиеся точкой в конце:

S1 -> S2.

Пример алгоритма Маркова №1

Задана строка из 0 и 1. Получить на выходе строку, в которой 1 заменены на 0, а 0 на 1. алгоритм задача сложность вычисление

; "*"-работник движется вдоль строки, и выполняет "работу" меняет 0->1, 1->0*0 > 1**1 > 0*; уничтожение "*"-работника, алгоритм завершается.* > .; ставим в самом начале звездочку-"работника" (замена пустой подстроки на *)> * Возьмем входную строку "1101".

Последовательно применяем подстановки. Первые три не подойдут, поскольку в строке нет "*". Но подойдет самая последняя подстановка, и звездочка будет добавлена.

Получим "*1101". Затем, последовательно применяя подстановки номер 1) и 2) будем получать: 0*101, 00*01, 001*1, 0010*. Далее применяем 3) завершающую подстановку. Звездочка в конце удаляется, получаем результат: 0010.

Пример алгоритма Маркова №2

Дано строковое представление числа в десятичной системе счисления, получить десятичное представление числа на 1 больше.

0# > 1.1# > 2.2# > 3.3# > 4.4# > 5.5# > 6.6# > 7.7# > 8.8# > 9.9# > #0*# > 1.*0 > 0**1 > 1**2 > 2**3 > 3**4 > 4**5 > 5**6 > 6**7 > 7**8 > 8**9 > 9* * > # > * Сперва срабатывает последняя подстановка, добавляющая "*" в начало числа.

Применяя подстановки *N > N*, она передвигается к концу числа. Когда звездочка дойдет до конца числа, она будет заменена на "#". Решетка (#) будет выполнять роль 1, которую мы прибавляем к числу. Посмотрим на первые 9 подстановок (N# > {N+1}., N=0..8).

Если последняя цифра числа от 0 до 8, то она увеличивается на 1 и алгоритм успешно завершается. Интереснее, когда в конце числа 9-ка, в этом случае срабатывает 9# > #0. Фактически это перенесение 1 в следующий разряд.