Введение в математическую логику и теорию алгоритмов

Лекция 12

Алексей Львович Семенов

План

- Вычислимость, разрешимость, породимость
- Сложность. Подход теории алгоритмов
- Время вычисления
- Реально решаемые задачи
- Задачи, решаемые перебором
- Универсальная переборная задача
- Естественные переборные задачи, их универсальность
- Проблема перебора

Переход к следующему в ансамбле

- Свойство всякого ансамбля
- Существует объект I (начальный) и действие S (следование), для которых {I, S(I), SS(I)…} – это весь ансамбль.
- **Задачи** построения функции следования.
  - Слова в унарном алфавите
  - Двоичный алфавит
  - Произвольный алфавит
  - Цепочки слов
  - Списки

Кодирование

- Кодирование элементов ансамбля словами в двоичном алфавите.
- **Задача**. Действие, задающее взаимно-однозначное соответствие (обратное – тоже действие) для ансамблей:
  - слов в любом алфавите,
  - цепочек слов,
  - списков.
- Тезис Поста. Тезис Черча – для кодов множеств в произвольных ансамблях.

Перечислимость. Эквивалентные определения

- Перечислимое множество – это либо пустое множество, либо множество значений какой-нибудь всюду определенной вычислимой функции.
- **Задача**. = области определения вычислимой функции
- Идеи:
  - Множество значений => область определения

  Для всякого исходного данного ждем, пока оно появится как значение, последовательно перебирая все аргументы для перечисления (действие следования). Если дождемся…
  - Область определения => множество значений всюду определенной функции

  Исходное данное рассматривать как пару: <Исходное данное, Число шагов> (если не пара – …). Если работа завершилась за это число шагов…
- **Задача**. = множеству значений вычислимой функции

Перечислимость и разрешимость

Теорема. Множество разрешимо тогда и только тогда, когда оно и его дополнение перечислимы.

- Д. Перечислимость – область определения
- Пусть множество разрешимо. Нужно в дополнении сделать функцию неопределенной.
- Пусть множество и его дополнение перечислимы.
- Идея: запустить *параллельно* функции, определенные на множестве и на дополнении.
- Уточнение. Делать в цикле *последовательно* по одному шагу одного и другого алгоритма. Когда один из них закончит работу, ответ есть.

Задача. Довершить рассуждения. В каком ансамбле?

Задача. Привести пример перечислимого неразрешимого множества (была функция, которую нельзя доопределить до всюду определенной вычислимой).

Сведение вычислимости к перечислимости

Задача. Функция вычислима <=> ее график перечислим.

Перечислимость ⇔ породимость

Перечислимость => породимость

- Проверка создания – действие преобразования

Породимость => перечислимость

- Всякое исходное данное – вывод.
- Результат

Функция вычислима <=> ее график породим

Перечислимость и логика отношений

- Множество общезначимых формул – область определения вычислимой функции
- Породимость – исчисление логики отношений

Сложность. Подход теории алгоритмов

Сложность вычислений

- О. *Сложность (временнАя) вычисления* – это число шагов вычисляющего алгоритма.
- Какими могут быть отдельные шаги? Какова «модель вычислений»?
- Например, алгоритмы Маркова или интуитивно представляемый компьютер.
- При конкретном исходном данном (или их конечном количестве) можно «запомнить ответ» и мгновенно его «выдать».
- Вопрос, есть ли алгоритм, быстро решающий данную задачу для заданного конечного множества исходных данных, имеет тривиальный ответ ДА.
- Поэтому, обычно говорят об асимптотическом поведении сложности вычисления, и это почти всегда оказывается осмысленным.

Реально решаемая задача

– Сложность вычисления ограничена полиномом от размера исходного данного (длины двоичного слова). Класс P.

• В реальных задачах коэффициенты и степени полиномов оказываются «небольшими».

• Бывает, что алгоритм очень просто описывается, но требует для своего выполнения много времени. Часто «много» означает экспоненту от размера исходного данного или еще больше.

Задачи, решаемые перебором

- Типичная ситуация (считаем, что объекты – двоичные слова, и их размер – это длина):

Задача о рюкзаке. Дано n+1 натуральное число: $a_0, a_1, \ldots, a_{n-1}, b$, можно ли составить из a_i сумму, равную b? (Каждое a_i разрешается брать не более одного раза.)

- Если брать только пары a_i => полиномиальное время – число пар квадратичное, умножить на время проверки, и т. д.
- Берутся не пары, а подмножества – время перебора экспоненциально.
- **23, 11, 44, 29, 18, 32, 19, 35, 67** – из этих чисел требуется составить сумму, равную **100**.
- **?**
- **Давайте попробуем.**

11, 18, 19, 23, 29

Алгоритмы для функций и отношений

- Задача здесь состояла в построении быстрого алгоритма для поиска подмножества индексов.
- Можно, однако, спрашивать о быстром алгоритме выяснения, есть ли такое подмножество.
- **Задача**. Имея такой общий быстрый (полиномиальный) алгоритм, построить алгоритм, который будет выдавать множество индексов.
- Мы будем рассматривать алгоритмы для отношений (алгоритмы распознавания).

Задачи, решаемые перебором

Выполнимость. Дана формула логики высказываний. Выполнима ли она?

- Можно выяснить, перебирая все возможные наборы значений логических имен.
- Разных имен в формуле может быть, по порядку, например, корень квадратный от размера формулы.
- Экспонента может быть не от размера, а от корня квадратного от размера, но это не очень помогает.

Задачи, решаемые перебором

Общая формулировка

• Дано двуместное отношение R(x,y), где x – исходное данное, y – перебираемое (подсказка, подтверждение).

- ***Задача***: выяснить по данному x, существует ли y, для которого R(x,y):

$$\exists y\ R(x,y),$$

причем:

  – размер y ограничен заданным полиномом от размера x,

  – сложность вычисления R(x,y) ограничена полиномом от размера x.

Отношение R и ограничивающие полиномы фиксированы для данной задачи.

• Задача о рюкзаке (номера) и Выполнимость (цепочка) – таковы.

- NP – Класс всех задач, решаемых перебором (N – недетерминированность).

Универсальная переборная задача

- **Универсальный алгоритм** p_0
- получает на вход x, второй аргумент отношения – y, описание конкретного алгоритма C и применяет C к x и y.
- Время работы p_0 не сильно отличается от времени работы C.
- Универсальный алгоритм p_0 задает отношение $U_0(z,y)$ такое, что если p – алгоритм, то $U_0(<x,p>,y)$ есть результат применения p к <x, y>. Если первый аргумент не оказывается парой такого вида, то отношение U_0 ложно.
- Отношение U_0 может и не вычисляться за полиномиальное время: для разных p полиномы, ограничивающие время работы и длину y, могут иметь разную степень, как функции длины x.

Универсальная переборная задача

- Модификация: отношение U, задаваемое, как

  U(<x, p, ℓ >, y), где алгоритм, вычисляющий U, работает так же, как алгоритм с описанием p,
- но при этом ограничивает время работы этого алгоритма p длиной слова – третьего элемента тройки.
  (Если первый аргумент U – не такая тройка, то U – Л, если p не завершает работу за данное время, тоже Л.)
- U имеет полиномиальную сложность.
- ∃yU(z,y) – *универсальная переборная задача.*

Универсальность

- Предположим, что мы нашли алгоритм, позволяющий вычислять отношение $\exists y U(z,y)$ за **полиномиальное** время.
- Этот алгоритм:
- НЕ перебирает y и
- НЕ применяет p_0,
- а делает что-то совсем другое.

- Тогда мы сможем и любую переборную задачу, заданную алгоритмом p и соответствующими полиномиальными ограничениями, решать за полиномиальное время, сооружая каждый раз тройку $< x, p, \ell >$, (и при этом не имитируя работу алгоритма с описанием p и не перебирая y).

Естественные переборные задачи

- Предположим, что мы можем решать задачу о рюкзаке (в формулировке «найдется ли…») или “Выполнимость” за полиномиальное время.
- Поможет ли это в решении других переборных задач?
- Да.

Сведение к выполнимости

- Пусть имеется **Задача**: $\exists y\ R(x,y)$, решаемая перебором, и отношение R задается алгоритмом p.
- Будем считать, что наш алгоритм p – это алгоритм Маркова.
- Построим исходное данное для задачи выполнимости, то есть формулу логики высказываний Ф.

Моделирование работы алгоритма р

с помощью выполнимости формул логики высказываний.

Первый элемент построения

• Все слова кодируем в двоичном алфавите (может оказаться удобным иметь коды равной длины для всех букв).

• Фиксируем натуральное число *n* и слово

$\alpha = \alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ в алфавите **B**;

Строим формулу Ф, куда входят имена $A = A_0, A_1, \ldots, A_{n-1}$,

• $(0, A_i) = \neg A_i$; $(1, A_i) = A_i$

$\Phi = (\alpha, A) = ((\alpha_0, A_0) \wedge (\alpha_1, A_1) \wedge \ldots \wedge (\alpha_{n-1}, A_{n-1})) = И \leftrightarrow$

«Значения имен цепочки А составляют слово α»

Длина Ф меньше, чем (10 длин α)·log (дл α) (индексы)

Моделирование работы алгоритма p

с помощью выполнимости формул логики высказываний

Мы уже знаем алгоритм и x, но, конечно, не знаем y

Общая схема:

Существование вычисления ⇔ существование выполняющего набора значений логических имен.

Вычисление – цепочка слов. Число слов в цепочке (число шагов алгоритма) ограничено полиномом от длины входа. Максимальная длина слова в цепочке ограничена полиномом (тем же, умноженным на константу).

Можно уложить все вычисление в прямоугольник (матрицу). В каждой строке матрицы будет двоичный код (при естественной кодировке) очередного элемента вычисления.

В самой верхней строчке (соответствующей началу вычисления) будет двоичный код пары <x,y>, сначала код x, потом код y.

Моделирование работы алгоритма p

с помощью выполнимости формул логики высказываний

Какая формула описывает вычисление?

Конъюнкция формул, относящихся к строкам:

Нулевая строка начинается с кода x.

Для каждого j от 0 до длины вычисления n:

j-ая строка или является кодом слова 1, тогда j+1-я с ней совпадает, или j+1-я строка получается из j–ой в соответствии с алгоритмом

n+1-ая строка есть код 1.

Самое сложное – написать формулу для перехода.

Давайте строить вспомогательные формулы.

Будем оценивать их размер.

Если для описания правильности перехода нам понадобится формула экспоненциальной длины, то наша попытка моделирования закончилась неудачей.

Слово $\beta = \beta_0, \beta_1, \ldots, \beta_{k-1}$ в алфавите **B**;

«Значения $A = A_0, A_1, \ldots, A_{n-1}$ составляют слово, в которое начиная с i–го места входит β»

$\Phi_i = ((\beta_0, A_i) \wedge (\beta_1, A_{i+1}) \wedge \ldots \wedge (\beta_{k-1}, A_{i+k-1}))$ – И

Длина Ф меньше, чем (10 длин β) ▪ log (дл α).

«Слово β входит в A: $\vee \Phi_i$, i= 0… n-I»

Длина формулы умножается на длину α

«Слово β входит в A»: дизъюнкция по местам

«Первое вождение слова β в A начинается с i–го места»:

Конъюнкция отрицаний по всем предшествующим местам и …

Моделирование работы алгоритма p

«Слова A и B совпадают до i–го места», слова A и B неизвестны, известны их длины и i.

«Слово A, начиная с (i+k)-го места, совпадает со словом B, начиная с (i+m)-го места.»

«Слово B получается из слова A заменой в нем вхождения слова β, начиная с i–го места, на слово γ» известны I, β и γ.

«Слово B получается из слова A заменой в нем первого вождения слова β, начиная с i–го места, на слово γ.»

«Слово B получается из слова A за один не последний шаг применения алгоритма p.»

Построение формулы Ф

- Пусть n – длина исходного данного.
- Длина вычисления – ограничена полиномом от n (= полиному).
- Длина промежуточных результатов – ограничена полиномом от n (= полиному).
- Сопоставим каждому моменту вычисления и каждому номеру символа в слове (промежуточном результате) логическое имя.
- Число логических имен – произведение полиномов.
- Формула строится для заданного алгоритма и исходного данного и означает:
- «Для заданного алгоритма и исходного данного значения имен описывают вычисление, заканчивающееся кодом 1.»
- Выполнимость формулы означает существование подсказки и вычисления на ней.
- **Задача**. Выписать более детально построение формулы и оценить ее размер.

Построение формулы Ф (повторение)

Конъюнкция условий на:

- Первое слово

  – первым элементом пары является конкретное слово x: Конъюнкция утверждений $A_i \equiv 0$ или $A_i \equiv 1$, фиксирующих значения x. (На часть, соответствующую y, никаких ограничений.) Длина – не более квадрата (число членов конъюнкции – длина x, индексы, скобки…)

- Каждые два последовательных слова: P и Q.

  «Слово B получается из слова A за один не последний шаг применения фиксированного алгоритма.»

- Последнее слово – это код 1, и нельзя применить ни одно из правил или на предпоследнем шаге применялось заключительное правило.

Проблема перебора

Доказать, что $P \neq NP$

Проблема равенства классов *P* и *NP* входит [первой] в семь задач тысячелетия, за решение которой Математический институт Клэя назначил премию в миллион долларов США.
– Википедия

А. Н. Колмогоров, Л. Левин, Л. Хачиян, А. Разборов