Компьютерная алгебра

(курс лекций)

Игорь Алексеевич Малышев

Computer.Algebra@yandex.ru

КСПТ

КОМПЬЮТЕРНЫЕ СИСТЕМЫ И ПРОГРАММНЫЕ ТЕХНОЛОГИИ

Лекция 2

Компьютерная обработка информации: модели, методы, средства

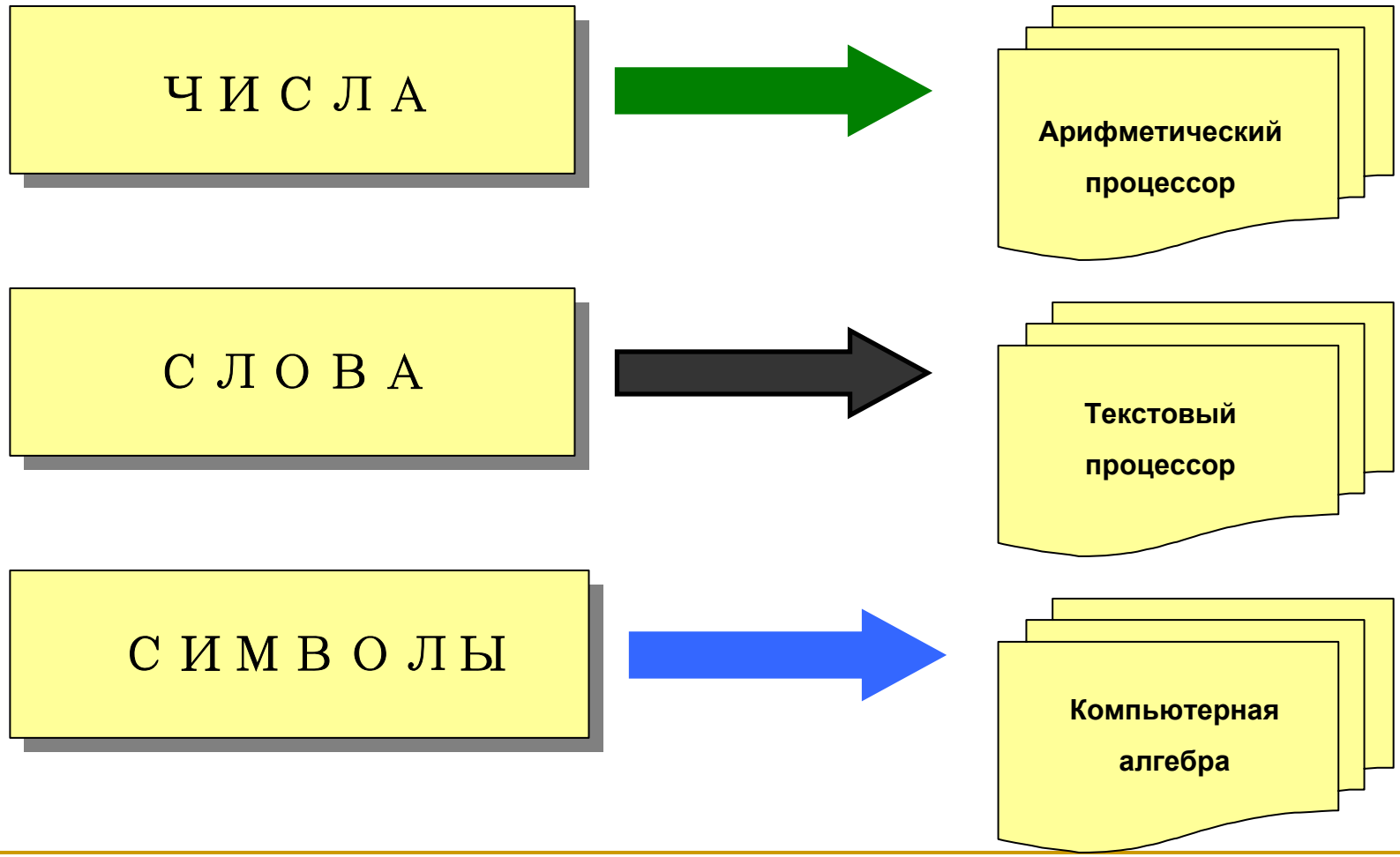
Содержание лекции

- Информационные объекты
- Компьютерная алгебра и численный анализ
- Элементы теории сложности алгоритмов

План лекции: тема подраздела

- **Информационные объекты**
- Компьютерная алгебра и численный анализ
- Элементы теории сложности алгоритмов

Информационные объекты: эволюция абстракций



Информационные объекты: символьная форма

Символы:

- Лингвистические символы (буквы, иероглифы, разделители и т.п.);
- Графические символы (геометрические фигуры, схемы, диаграммы и т.п.);
- Математические символы (в основном, точные числа и алгебраические выражения).

Информационные объекты: от математики к алгоритмике

Компьютерная алгебра – это наука об эффективных алгоритмах вычислений математических объектов.

План лекции: тема подраздела

- Информационные объекты
- **Компьютерная алгебра и численный анализ**
- Элементы теории сложности алгоритмов

Компьютерная алгебра и численный анализ

Типовая математическая задача.

Вычислить определённый интеграл:

$$r = \int_0^1 arctg\, x\, dx$$

Численное решение:

Используя численное интегрирование, получим приближённое значение ***r*** в виде числа с плавающей точкой двойной точности:

r = 0.4388245731174756

Достоверность решения ?
Сомнительна.
Информативность решения ?
Минимальна.

Компьютерная алгебра и численный анализ

Аналитическое решение (СКА «Maple»).

$$r = \int_0^1 arctg\, x\, dx$$

Значение неопределённого интеграла:
int(arctan(x),x);

$$x\arctan(x) - \frac{1}{2}\ln(1+x^2)$$

Значение ***r*** в аналитическом виде:
r := int(arctan(x),x=0..1);

$$r := \frac{1}{4}\pi - \frac{1}{2}\ln(2)$$

Численное значение ***r*** с любой точностью (например, 50 десятичных цифр после запятой):
evalf(r, 50);

.43882457311747565490704478509078743701154228266365

Аналитическое решение ?
Достоверно и максимально информативно.

Численное решение ?
Имеет переменную достоверность.

Компьютерная алгебра и численный анализ

Две основные проблемы Computer Science:

(1) проблема **точности** представления континуальных множеств с помощью конечных образов;

(2) проблема **сложности** реализации вычислительных алгоритмов при ограничениях на ресурсы.

Компьютерная алгебра и численный анализ

Численный анализ:

Решение проблемы точности:

приближение
бесконечного множества вещественных чисел
с помощью конечного множества чисел с плавающей точкой.

Решение проблемы сложности:

переопределение
математического понятия «сложность алгоритма»
с помощью компьютерного понятия «сложность программы».

Компьютерная алгебра и численный анализ

Численный анализ:

Система чисел F
с плавающей точкой:

$$f = \pm\left(\frac{d_1}{B} + \frac{d_2}{B^2} + \ldots + \frac{d_m}{B^m}\right) B^e$$

B – основание счисления;

m – точность мантиссы;

$[L, U]$ – область значений экспоненты.

$$d_i, i = 1, 2, \ldots m;$$

$$0 \le d_i \le B - 1;$$

$$L \le e \le U;$$

$$d_1 \ne 0 \text{ для } f \in F, f \ne 0$$

Все параметры явно зависят от компьютера.

Компьютерная алгебра и численный анализ

Численный анализ:

Машинные числа распределены равномерно не на всей области значений, а только между последовательными степенями B.

Компьютерная алгебра и численный анализ

Численный анализ:

Пример. 33-точечное множество F:

$$B = 2, m = 3, L = -1, U = 2$$

Операция сложения:

$$\frac{5}{4} + \left(\frac{3}{8} + \frac{3}{8} \right) = 2 \text{ , но } \left(\frac{5}{4} + \frac{3}{8} \right) + \frac{3}{8} \neq 2,$$

$$\text{т.к. } S = \left(\frac{5}{4} + \frac{3}{8} \right) \notin F, S \approx \frac{3}{2} \in F \text{ или } S \approx \frac{7}{4} \in F$$

Компьютерная алгебра и численный анализ

Численный анализ (выводы):

(1) В множестве F существует ровно

$$2\left(B-1\right)B^{m-1}\left(U-L+1\right)+1$$

(нормализованных чисел с плавающей точкой, включая нуль).

(2) Множество F не является бесконечным.

(3) Множество F не является непрерывным.

(4) Приближение истинных результатов вычислений к представимым числам порождает ошибки округления.

(5) Арифметические операции (+, *) не ассоциативны и не дистрибутивны.

Компьютерная алгебра и численный анализ

Компьютерная алгебра:

Решение проблемы точности:

представление числовой информации
с помощью целых и рациональных чисел
произвольной точности.

Решение проблемы сложности:

измерение
сложности алгоритмов
с помощью абстрактных унифицированных метрик.

Компьютерная алгебра и численный анализ

Компьютерная алгебра:

1-я сопутствующая проблема –

разнообразие типов данных.

Решение: переменный тип представления информации.

Логика преобразования типов представлений информации
переопределяется по мере необходимости
с помощью операторов языка символьных вычислений.
(В численном анализе все типы представлений информации неизменны и соответствуют машинным представлениям данных).

2-я сопутствующая проблема –

разнообразие математических операций.

Решение: переменная арность математических операций обработки данных.

Функциональный тип (сложение, умножение, максимум, медиана и т.п.) и арность (2, 3, и т.д.) операций обработки данных определяются независимо. Функциональный тип постоянен. Базовая арность равна 2. Принцип реализации n-арных (n > 2) операций аналогичен правилам выполнения арифметических операций над числами произвольной точности.
(В численном анализе все операции обработки данных являются либо унарными, либо бинарными и полностью соответствуют машинным командам).

Компьютерная алгебра и численный анализ

Компьютерная алгебра лучше численного анализа ?

Контраргументы:

1) Точные аналитические вычисления (при прочих равных условиях) выполняются медленнее (часто - значительно медленнее (в десятки раз)), чем численные (приближённые).
2) Далеко не всякая математическая задача имеет аналитическое решение – потенциальное (определяемое существующими математическими формализмами) или реальное (зависящее от фактической размерности задачи).
3) Значительное количество практически важных задач пока не формализованы настолько, чтобы решаться аналитически.

План лекции: тема подраздела

- Информационные объекты
- Компьютерная алгебра и численный анализ
- **Элементы теории сложности алгоритмов**

Элементы теории сложности алгоритмов

Классы задач обработки информации :

- Алгоритмически неразрешимые
- Задачи, алгоритмическая неразрешимость которых не доказана
- Алгоритмически разрешимые

Элементы теории сложности алгоритмов

Алгоритмически неразрешимые задачи :

Существование таких задач – следствие двух теорем К. Гёделя (1931 г.) о неполноте символических логик:

Теорема 1. Если формальная арифметика непротиворечива, то в ней существует не выводимая и неопровержимая формула.

Теорема 2. Если формальная арифметика непротиворечива, то в ней не выводима некоторая формула, содержательно утверждающая непротиворечивость этой арифметики.

«Эталон» алгоритмически неразрешимой задачи – задача останова машины А.Тьюринга (за **конечное** количество шагов).

Теорема (А. Тьюринг, 1936). Не существует алгоритма, позволяющего определить по описаниям какого-либо алгоритма и его исходных данных (в виде машины Тьюринга) останавливается такой алгоритм или работает бесконечно.

Элементы теории сложности алгоритмов

Аспекты алгоритмической неразрешимости - 1:

(1) отсутствие общего метода решения задачи

Проблема 1. Распределение девяток в записи числа π.

Определим функцию f(n) = i, где n – количество девяток подряд в десятичной записи числа π, а i – номер самой левой девятки из n девяток подряд: π = 3,141592… f(1) = 5.

Задача состоит в вычислении функции f(n) для произвольно заданного n.

Поскольку число π является иррациональным и трансцендентным, то мы не знаем никакой информации о распределении девяток (равно как и любых других цифр) в десятичной записи числа. Вычисление f(n) связано с вычислением последующих цифр в разложении π , до тех пор, пока мы не обнаружим n девяток подряд, однако у нас нет общего метода вычисления f(n), поэтому для некоторых n вычисления могут продолжаться бесконечно – мы даже не знаем в принципе (по природе числа π) существует ли решение для всех n.

Проблема 2. Вычисление совершенных чисел.

Совершенные числа – это числа, которые равны сумме своих делителей, например: 28 = 1+2+4+7+14.

Определим функцию S(n) = n-ое по счёту совершенное число и поставим задачу вычисления S(n) по произвольно заданному n.

Нет общего метода вычисления совершенных чисел, мы даже не знаем, множество совершенных чисел конечно или счетно, поэтому наш алгоритм должен перебирать все числа подряд, проверяя их на совершенность. Отсутствие общего метода решения не позволяет ответить на вопрос о останове алгоритма. Если мы проверили M чисел при поиске n-ого совершенного числа – означает ли это, что его вообще не существует?

Проблема 3. Десятая проблема Д. Гильберта.

Пусть задан многочлен n-ой степени с целыми коэффициентами – P, существует ли алгоритм, который определяет, имеет ли уравнение P=0 решение в целых числах?

Ю.В. Матиясевич показал, что такого алгоритма не существует, т.е. отсутствует общий метод определения целых корней уравнения P=0 по его целочисленным коэффициентам.

Элементы теории сложности алгоритмов

Аспекты алгоритмической неразрешимости - 2:

(2) информационная неопределённость задачи

Проблема 4. Позиционирование машины Поста на последнюю помеченную ячейку.

Пусть на ленте машины Поста заданы наборы помеченных ячеек (кортежи) произвольной длины с произвольными расстояниями между кортежами и головка находится у самой левой помеченной ячейки. Задача состоит в установке головки на самую правую помеченную ячейку последнего кортежа.

Попытка построения алгоритма, решающего эту задачу приводит к необходимости ответа на вопрос – когда после обнаружения конца кортежа мы сдвинулись вправо по пустым ячейкам на M позиций и не обнаружили начало следующего кортежа – больше на ленте кортежей нет или они есть где-то правее? Информационная неопределенность задачи состоит в отсутствии информации либо о количестве кортежей на ленте, либо о максимальном расстоянии между кортежами – при наличии такой информации (при разрешении информационной неопределенности) задача становится алгоритмически разрешимой.

(3) логическая неразрешимость (в смысле теорем К. Гёделя о неполноте)

Проблема 5. Проблема «останова» (см. теорему А. Тьюринга).

Проблема 6. Проблема эквивалентности алгоритмов.

По двум произвольным заданным алгоритмам (например, по двум машинам Тьюринга) определить, будут ли они выдавать одинаковые выходные результаты на любых исходных данных.

Проблема 7. Проблема тотальности.

По произвольному заданному алгоритму определить, будет ли он останавливаться на всех возможных наборах исходных данных. Другая формулировка этой задачи – является ли частичный алгоритм P всюду определённым?

Элементы теории сложности алгоритмов

Алгоритмически неразрешимые задачи компьютерной алгебры :

- Проблема тождества для полугрупп;
- Неразрешимость системы диофантовых уравнений;
- Проблема сравнения значений функций;
- Проблема аналитической интегрируемости некоторых выражений;
- и др. задачи.

Элементы теории сложности алгоритмов

Если алгоритмическая разрешимость – открытая проблема ?

Такие задачи, как правило, происходят из алгоритмически неразрешимых задач. К ним относятся, например, следующие:

- Аналог 10-й проблемы Д.Гильберта для уравнений 3-й степени;
- Аналог 10-й проблемы Д.Гильберта для уравнений в рациональных числах.

Элементы теории сложности алгоритмов

Алгоритмически разрешимые задачи :

Известно, что алгоритмически неразрешимых (и приравненных к ним) задач бесконечно больше, чем алгоритмически разрешимых.

Проблема алгоритмически разрешимых задач – трудоёмкость алгоритмов их решения.

«Спасибо тебе, Господи, что ты создал всё нужное – нетрудным, а всё трудное - ненужным».

(Георгий Сковорода, украинский философ)

Элементы теории сложности алгоритмов

Понятие трудоёмкости алгоритма

- Трудоёмкость алгоритма –

  количество ресурсов (вычислительной) машины для реализации алгоритма.
- Типы вычислительных ресурсов –

  --- ёмкостные (объём памяти машины: количество ячеек);

  --- временные (длительность работы машины: количество операций).
- Оценка трудоёмкости –

  функция от свойств входных данных алгоритма.

Элементы теории сложности алгоритмов

Классы трудоёмкости алгоритмов :

- Количественно-зависимые.

  Трудоёмкость зависит только от размерности входных данных.

  (стандартные операции над массивами и матрицами)
- Параметрически-зависимые.

  Трудоёмкость зависит только от значений входных данных (при этом размерность - постоянна).

  (приближение стандартных функций с помощью степенных рядов)
- Порядково-зависимые.

  Трудоёмкость зависит от порядка расположения входных данных.

  (сортировка, поиск минимума-максимума в массиве)
- Количественно-параметрические.

  Трудоёмкость зависит и от размерности, и от значений входных данных.

  (численный алгоритм: параметрически-зависимый внешний цикл по точности, количественно-зависимый внутренний цикл по размерности)

Понятие сложности алгоритма

Проблема определения трудоёмкости алгоритма – неоднозначность оценок :

(1) оценки в лучшем (минимальная трудоёмкость), худшем (максимальная трудоёмкость) и среднем случаях;

(2) оценки, приведённые к разным входам алгоритма (если таких входов несколько).

Решение проблемы –
введение асимптотических оценок трудоёмкости алгоритма (сложность алгоритма).

Типы асимптотических оценок :

- оценка сверху (O большое – O (n));
- оценка снизу (Ω - Ω (n)).

Элементы теории сложности алгоритмов

Асимптотический анализ

Определение (свойство доминантности функций).

Пусть f (s) и g (s) – положительные функции положительного аргумента (обычно s ≥ 1, если s – количество входных данных алгоритма), s ∈ S. Будем говорить, что:

(1) функция f ***доминируется*** (мажорируется) функцией g (обозначение: f = O (g)), если существует положительное вещественное число c_1 и элемент $s_0 \in S$ такие, что | f (s) | ≤ c_1 | g (s) | для всех s > s_0;

(2) f доминирует g (обозначение: f = Ω (g)), если g доминируется f;

(3) функции f и g кодоминантны (обозначение: f ~ g), если f = O (g) и f = Ω ().

Определение (функция времени вычислений).

Пусть A – алгоритм и S – множество допустимых значений входа для A. Тогда целое число T_A (n) для n ∈ S – число базисных операций, выполняемых алгоритмом A при значении входа n, называется ***функцией времени вычислений***, ассоциированной с A и определённой на S.

Элементы теории сложности алгоритмов

Базисные операции алгоритмов :

- Простое присваивание: a <- b;
- Одномерная индексация a [i] :
  (адрес (a) + i *длина элемента);
- Арифметические операции: (*, /, -, +)
- Операции сравнения: a < b
- Логические операции:
  (e1) {or, and, not} (e2)
- Операция передачи управления

Элементы теории сложности алгоритмов

Оценки $T_A(n)$ для алгоритмических конструкций

Конструкция «Следование»

Оценка времени вычисления - сумма времён вычислений блоков, следующих друг за другом:

$T_{\text{«следование»}} = t_1 + \ldots + t_k$, где k – количество блоков.

Конструкция «Ветвление»

Оценка времени вычисления – свёртка по вероятностям переходов на блоки THEN и ELSE:

$T_{\text{«ветвление»}} = t_{THEN} * p + t_{ELSE} * (1 - p)$, где p – вероятность перехода на THEN

Конструкция «Цикл»

(цикл с параметром: *for i=1 to N do* **«тело цикла»)**

$T_{\text{«цикл»}} = 1 + 3 * N + N * t_{\text{«тело цикла»}}$

Элементы теории сложности алгоритмов

Примеры анализа T_A (n) для простых алгоритмов

Пример 1. Задача суммирования элементов квадратной матрицы.

```
SumM (A, n; Sum)
        Sum <- 0
        For i <- 1 to n
            For j <- 1 to n
                Sum <- Sum + A [ i, j ]
            end for
        end for
        Return (Sum)
End
```

Результаты анализа :

Количественно-зависимый алгоритм.

$T_{SumM} = 7 * n^2 + 4 * n + 2 = O(n^2)$

Элементы теории сложности алгоритмов

Примеры анализа T_A (n) для простых алгоритмов

Пример 2. Задача поиска максимума в массиве.

```
MaxS (S, n; Max)
          Max <- S [ 1 ]
          For i <- 2 to n
               if Max < S [ i ]
                  then Max <- S [ i ]
          end for
          Return (Max)
End
```

Результаты анализа :

Количественно-параметрический алгоритм. Дополнительный анализ:

(1) Лучший случай – максимальный элемент массива на первом месте.

$T_{MaxS} = 5 * n - 2 = O (n)$

(2) Худший случай – массив отсортирован по возрастанию.

$T_{MaxS} = 7 * n - 4 = O (n)$

(3) Средний случай – при равномерном распределении исходных данных вероятность расположения к-го элемента массива на определённом (последнем) месте равна 1/к.

$T_{MaxS} = 5 * n + 2 * \log (n) – 4 + 2 * \gamma = O (n)$, где $\gamma \approx 0.57$

Элементы теории сложности алгоритмов

Классы сложности алгоритмов - 1

(Алан Кобен, 1964; Джек Эдмондс, 1965)

Класс P – задачи с полиномиальной сложностью.

Задача называется полиномиальной, если существует константа k и алгоритм A, решающий задачу за $T_A(n) = O(n^k)$

(для большинства таких задач $k < 6$, это «практически разрешимые задачи»)

Класс NP – полиномиально проверяемые задачи.

Решение такой задачи может быть быстро (с помощью алгоритма полиномиальной сложности) проверено.

Открытая проблема теории сложности: P = NP ?

Т.е. можно ли все задачи, решение которых проверяется с полиномиальной сложностью, решить за полиномиальное время ?

Элементы теории сложности алгоритмов

Факторы, не учитываемые при анализе $T_A(n)$:

- **Неадекватность формальной системы записи алгоритма и реальной системы команд процессора;**
- **Наличие архитектурных особенностей, существенно влияющих на время выполнения алгоритма (программы) – конвейеры, кэширование, предвыборка команд и данных и т.п.;**
- **Различные времена выполнения реальных машинных команд;**
- **Различие времени выполнения одной команды в зависимости от значений операндов;**
- **Различные времена реального выполнения однородных команд в зависимости от типов данных;**
- **Неоднозначности компиляции исходного текста, обусловленные как алгоритмом самого компилятором, так и его настройками.**

Классы сложности алгоритмов - 2

Класс NPC – NP-полные задачи (Стефен Кук, 1971).

Задача называется NP-полной, если:

(1) она принадлежит к классу NP;

(2) к ней полиномиально сводятся все задачи из класса NP.

Полиномиальная сводимость одной задачи к другой означает существование алгоритма полиномиальной сложности, способного выполнить трансляцию описания исходной задачи на одном языке в эквивалентное ему описание на другом языке.

Большинство комбинаторных задач – это NP-полные задачи.

(Примеры. 1. Задача определения клики в графе. Клика – максимальный по количеству вершин полный подграф графа. 2. Задача о выполнимости булевой формулы).

Элементы теории сложности алгоритмов

Классы сложности алгоритмов - 3

Класс задач экспоненциальной сложности.

Функции времени вычисления для таких задач являются показательными от количества входных данных.

Замечательное неравенство (шкала сложности) :

$$O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n)$$

Общий вывод :

Вычисление является «лёгким» (эффективным), если мы имеем дело с полиномиальным по времени алгоритмом, и «сложным», если мы имеем дело с экспоненциальным по времени алгоритмом.

Спасибо за внимание !

Вопросы ?