Математические основы информатики

Лекция 1.

Сергей Леонидович Бабичев

Что такое программирование?

- Это набор каких-то текстов программ на компьютере?
- Это придумывание каких-то алгоритмов?
- Это реализация этих алгоритмов на каком-то языке программирования?

Мы, профессионалы, делим программистов на тех, кто «рисует» интерфейсы программ, *frontend* и тех, кто реализует невидимые для пользователя движущие силы, *backend*.

Чему учат курсы по программированию?

- Как нарисовать пользовательский интерфейс.
- Какие кнопки нажимать в данной программе.
- Какую функцию языка использовать в том или ином случае.

Чему не учат курсы про программированию?

- Как создавать алгоритмы.
- Как обосновывать ваш выбор.
- Как доказывать корректность алгоритма.

Почему? Для этого нужна достаточно серьёзная математическая основа.

Кто создаёт новые алгоритмы?

Программист? Нет, математик, который программирует.

Только математика позволяет придумывать и обосновывать интересные, полезные и эффективные алгоритмы.

Наша деятельность: первый семестр

Наш курс — четыре семестра.

- Первый семестр — математические основы программирования.
- К некоторым темам, которые мы будем изучать, вы вернётесь через год или два на более глубоком уровне.

Наша деятельность: второй семестр

- Второй семестр — простые алгоритмы и структуры данных. Стек, очередь, вектор, кучи (двоичная, биномиальная, косая и фибоначчиева); сортировки, поиск, деревья поиска (splay, AVL, декартово, B-дерево); обработка запросов: хеширование и хеш-таблицы, фильтр Блума, алгоритм Карпа-Рабина, деревья отрезков, деревья Фенвика, разреженные таблицы. Различные техники оценки временной сложности алгоритмов.

Наша деятельность: третий семестр

- Третий семестр — динамическое программирование: уравнение Беллмана, многомерные варианты, использование дерева отрезков и дерева Фенвика, динамическое программирование по битовым маскам и по контуру.
- Графы, их обход графов. Алгоритмы Косарайю и Тарджана. Конденсация графа. Мосты и точки сочленения. Отношение эквивалентности R. Рёберная двусвязность. $2SAT$. Поиск кратчайшего расстояния в графах. Алгоритмы Дейкстры, $A*$, Флойда-Уоршалла, Джонсона, Форда-Беллмана. Матрица транзитивного замыкания. Алгоритм Прима. Лемма о безопасном ребре. Система непересекающихся множеств. Алгоритм Краскала, Борувки. Паросочетания в произвольном графе. Двудольные графы. Понятие увеличивающего пути. Теорема Бержа. Потоки в графах. Алгоритмы Форда-Фалкерсона, Эдмондса-Карпа. Слоистая сеть. Блокирующий поток. Алгоритм Диница. Определение центроида в дереве. Лемма о количестве центроидов. Изоморфизм графов. Задача LCA. Алгоритм Фарах-Колтона и Бендера. Задача RMQ. Heawy-light декомпозиция. Тяжёлые и лёгкие рёбра. Центроидная декомпозиция.

Наша деятельность: четвёртый семестр

- Алгоритмы теории чисел. Быстрая факторизация. Преобразования Фурье. Криптографические алгоритмы. Алгоритмы на строках: Кнута-Мориса-Пратта, Карпа-Рабина, Ахо-Корасик. Суффиксные массивы, LCP, алгоритм Касаи. Суффиксные деревья. Суффиксные автоматы. Алгоритм Укконена. Алгоритмы, применяемые для сжатия (компрессии) текста без потерь: Huffman, LZ78, LZ77, BWT, context. Алгоритмы вычислительной геометрии: Грэхема, Джарвиса, Эндрю, вращающихся калиперов, триангуляции, кратчайших расстояний. Сумма Минковского. Диаграмма Вороного. Приближённые и эвристические методам решения NP-сложных и обратных задач. Метод имитации отжига, генетические алгоритмы.

Причём здесь математика?

- Всё это невозможно без развитого математического аппарата.
- За весь курс у вас будет более сотни алгоритмов и ещё больше задач для их реализации.
- В этом семестре у нас математика, связанная с программированием.

Что же нам предстоит

- Основы теории множеств и логики.
- Основы целочисленной арифметики и побитовых операций.
- Основы комбинаторики.
- Рекурренты и индукция.
- Целочисленные функции и их свойства.
- Асимптотические отношения и операции над ними.
- Сложность вычислений.
- Основы теории чисел.
- Основы нецелочисленной арифметики.
- Основы теории вероятности.
- Основы теории графов.
- Основы теории игр.
- Основы вычислительной геометрии.
- Введение в корректность алгоритмов.

Несколько примеров.

Несколько слов об алгоритмах

Что такое алгоритм? Неформально:
Определение: Алгоритм — последовательность команд для *исполнителя*, обладающая свойствами

- **полезности**, умения решать поставленную задачу;
- **детерминированности**, строгой определённости каждого шага во всех возможных ситуациях;
- **конечности**, способности завершаться для любого множества входных данных;
- **массовости**, применимости к разнообразным входным данным;
- **корректности**, получения верных результатов для всех допустимых входных данных.

Исполнение алгоритмов

- Алгоритм *исполняется* или *вычисляется* над некоторыми *входными данными* и требует для исполнения некоторых *ресурсов*.
- Алгоритм становится *программой*, когда он записан на некотором формальном *языке программирования*.
- При исполнении алгоритма производятся *действия* или *операции*, которые могут делиться на более мелкие, что приводит к *элементарным действиям*.
- Степень элементарности операций различна для различных исполнителей.
- Например, для исполнителя `python` целое число — элементарное данное, умножение любых целых чисел — элементарная операция.
- Для исполнителя *процессор Intel/AMD* произвольное целое число не является элементарным типом данных и операция умножения таких чисел не является элементарной операцией.
- Почему? Сейчас увидим.

Числа и информация

- Те компьютеры, с которыми мы сталкиваемся, представляют числа в двоичной системе счисления.
- Это значит, что в представлении чисел присутствуют на машинном уровне только нули и единицы.
- Это хорошо: представление чисел, и операции над ними, подчиняются математической логике.
- У вас будет отдельный предмет «Логика»
- Мы затронем только те разделы логики, которые непосредственно связаны либо с представлением чисел, либо с доказательством корректности алгоритмов посредством цепочки логических выражений.

Числа, комбинаторика, теория информации

- Двоичное представление приводит к комбинаторике.
- Комбинаторика даёт ответ на вопрос: сколько существует чисел, которые можно записать 8, 16 и так далее двоичными разрядами?
- Теория информации, определяет информационную ёмкость числа с точки зрения теории вероятности.

$$I = \log_2 N, \tag{1}$$

  где I — количество информации в *битах*, необходимое для представления целого числа в диапазоне $[0; N)$.
- Я не забыл взять функцию, которая называется `floor` и обозначается $\lfloor x \rfloor$.
- В четвёртом семестре мы выясним, что информацию можно измерять и кодировать и дробным количеством бит (*арифметическое кодирование*) и мы тогда напишем нетривиальный архиватор.

Вероятность и комбинаторика

- Другая формула:

$$I = -\log_2 p, \tag{2}$$

где p — вероятность события «выбор данного числа из множества». Здесь у нас появляется теория вероятностей, теория множеств и опять комбинаторика: вероятность какого-то события есть отношение числа успешных событий ко всем при условии, что все события равновероятны.

Информационная ёмкость чисел

- Можно ли ответить на вопрос: какая информационная ёмкость числа 1?
- А числа 100?
- Чтобы ответить на него нужно знать количество чисел, из которых мы выбираем.
- Если таких чисел конечное множество, то и информационная ёмкость конечна.
- Например, если в каком-то типе данных, обрабатываемом компьютером, можно закодировать числа от -128 до 127 включительно, то, при условии, что число x принадлежит этому диапазону, информационная ёмкость числа x будет равна 8 битам.

Числа Фибоначчи

$$\begin{cases} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2}, \text{ если } n > 1. \end{cases} \tag{3}$$

- Можно ли определить информационную ёмкость нулевого числа Фибоначчи? В двоичном представлении оно занимает ровно один бит.
- Из какого множества чисел мы будем выбирать этот ноль?
- Какая информационная ёмкость 1000-го Фибоначчи? Оно имеет 209 десятичных цифр в записи. Верно ли, что $I(F_{1000}) = 4 \times 209 = 836$ бит?
- Или $694 = \log_2 F_{1000}$ бита?
- Как разделять числа? А как кодировать разделители, если у нас в распоряжении только 0 и 1? Последовательностью 0 и 1? Это тоже какое-то двоичное число.
- Исполнитель `python` скрывает эту сложность. Исполнитель *процессор* её показывает.
- В математике мы имеем дело с *абстрактными* числами, а в алгоритмике мы должны иметь дело с *представлениями* чисел.

Исполнитель *центральный процессор*

Пока переключимся на операции с целыми числами и на исполнитель *центральный процессор*.

- Зависит ли время исполнения элементарных операций от размера операндов?
- Да, должно ведь зависеть.
- Все ли размеры операндов одинаково удобны для исполнителя?
- Нет. Операнды в 5, 17, 96 бит можно обрабатывать, но это — не элементарные операнды, и их не получится обрабатывать элементарным образом.
- К элементарным данным относятся 8, 16, 32 и 64-битные числа.

Чанки

- Выберем у исполнителя идеальную для него элементарную структуру данных, время обработки которой минимально, а количество обрабатываемых бит — максимально, назовём её *чанк*.
- Представлять числа произвольной длины достаточно просто: мы храним количество чанков в числе и сами чанки в виде какого-то массива.
- Назовём число, состоящее из n чанков $(n)-$числом.
- Мощность множества чисел, которые могут быть представлены каждым чанком, обозначим R.
- Тогда $(n)-$число x есть

$$x = a_{n-1}R^{n-1} + \cdots + a_1R^1 + a_0R^0, \tag{4}$$

где $a_i \in [0; R)$.
Так это же запись числа в позиционной системе счисления по основанию R.

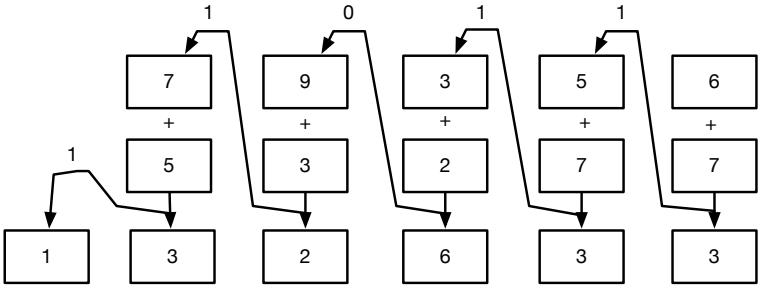
Объект *число*

- Мы создали неэлементарный объект *число* из элементарных объектов *чанки*.
- Осталось определить, насколько быстро можно производить операции над нашими *числами* с использованием элементарных операций над *чанками*.

Сложность алгоритма

- Термин *сложность алгоритма* не вполне однозначный.
- Для специалиста по схемотехнике сложность реализуемого в виде устройства алгоритма может определяться количеством микросхем и/или вспомогательных элементов монтажа. Его интересует *комбинационная сложность* алгоритма.
- Тех программистов, которым важно время написания программы, но не важно время её исполнения, интересует *описательная сложность* алгоритма. Не напрасно же мы наблюдаем успех `python`.
- Нас будет интересовать *вычислительная сложность* алгоритма, причём с разных сторон.
- Одна сторона — количество требуемых операций.
- Другая сторона — требуемые дополнительные ресурсы для хранения объектов, необходимых для исполнения.
- И эту сложность можно формализовать, введя один *главный параметр* алгоритма или несколько.
  Какова, в частности, сложность алгоритма сложения двух $(n)-$чисел?

Сложение двух $(n)-$чисел



- Если одно число короче другого, его дополним нулями слева.
- Тогда количество операций будет пропорционально главному параметру — длине числа.
- Мы потом будем называть буквой тета — $\Theta(N)$.
- Вычитание — аналогично. А вот что с умножением?

Умножение двух $(n)-$чисел

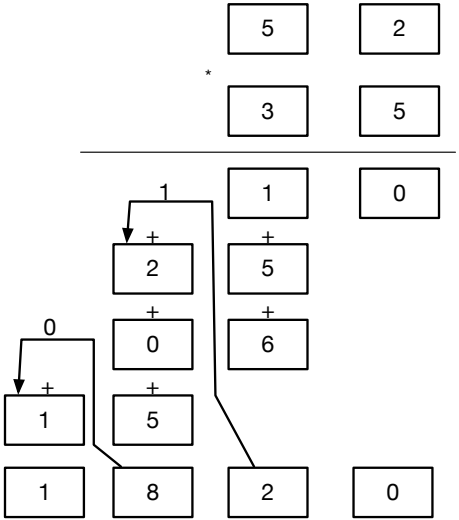


Рис.: Школьный алгоритм умножения

Алгоритм Карацубы

- Алгоритм Карацубы реализует принцип Цезаря — *разделяй и властвуй*.
- Для простоты давайте попробуем перемножить два $(2n)-$числа.
- $(n)-$числа можно очень быстро умножать их на R, основание системы счисления, приписав справа один нуль.
- Можно приписать сразу k нулей, это умножит число на R^k.
- Введём число $T = R^n$.
- Тогда любое $(2n)-$число X можно представить в виде суммы $Tx_u + x_l$. Это разложение имеет сложность $O(n)$, так как оно заключается просто в копировании соответствующих разрядов $(n)-$чисел.

$$N_1 = Tx_1 + y_1 \tag{5}$$
$$N_2 = Tx_2 + y_2 \tag{6}$$

Алгоритм Карацубы

- При умножении в столбик

$$N_1 \times N_2 = T^2 x_1 x_2 + T((x_1 y_2 + x_2 y_1) + y_1 y_2. \tag{7}$$

  Это — четыре операции умножения и три операции сложения.
- Алгоритм Карацубы находит произведение по другой формуле:

$$N_1 \times N_2 = T^2 x_1 x_2 + T((x_1 + y_1)(x_2 + y_2) - x_1 x_2 - y_1 y_2) + y_1 y_2. \tag{8}$$

- В её истинности нетрудно убедиться, просто раскрыв скобки.
- Мы уменьшили на единицу количество операций умножения, но сложений теперь стало на три больше.

Сложность алгоритма Карацубы

- Стал ли алгоритм иметь меньшую сложность?
- Проделаем несколько экспериментов чтобы определить количество умножений и сложений для больших чисел.
- Как определить сложность такого алгоритма? Можно попробовать выразить количество операций одного уровня через количество операций другого уровня. Обозначим количество операций умножения уровня k за m_k, а количество операций сложения — за a_k.
- Чтобы произвести одну операцию умножения, требуется провести три операции умножения и шесть операций сложения предыдущего уровня.
- Следующий уровень определяет числа, с количеством чанков в два раза большим, чем предыдущий уровень.
- Количество чанков равно 2^k, где k — номер уровня.
  Можно записать это так, первый элемент пары — число умножений, второй — число сложений:

$$m_k = (3m_{k-1}, 6a_{k-1}). \tag{9}$$

Рекурренты

- Как определить количество операций сложения в зависимости от количества операций предыдущего уровня?
- Чему равно a_k в зависимости от a_{k-1} и m_{k-1}? Перефразируем.
- Чтобы совершить операцию умножения чисел на уровне k нужно 3 операции умножения чисел уровня $k-1$ и 6 операций сложения чисел уровня $k-1$.
- Для сложения чисел на уровне k нужно в два раза больше операций, чем для такого же сложения, на уровне $k-1$. Таким образом,

$$a_k = (0m_{k-1}, 2a_{k-1}). \tag{10}$$

Всё это можно записать вместе:

$$\begin{cases} m_k &= (3m_{k-1}, 6a_{k-1}), \text{ если } k > 0; \\ a_k &= (0m_{k-1}, 2a_{k-1}), \text{ если } k > 0; \\ m_0 &= (1, 0); \\ a_0 &= (0, 1) \end{cases} \tag{11}$$

Исследуем рекурренту от двух переменных

- Найдём руками несколько первых членов этой рекуррентной последовательности.

$$
\begin{aligned}
(m_0, a_0) &= ((1,0),(0,1)) \\
(m_1, a_1) &= ((3,6),(0,2)) \\
(m_2, a_2) &= ((9,30),(0,4)) \\
(m_3, a_3) &= ((27,114),(0,8) \\
(m_4, a_4) &= ((81,390),(0,16) \\
&\dots \\
(m_{10}, a_{10}) &= ((59049,348150),(0,1024))
\end{aligned}
\tag{12}
$$

Исследуем рекурренту

- Для чисел с 1024 чанками количество операций умножения оказалось 59049, а операций сложения — 348150.
- Операция целочисленного умножения в несколько десятков медленнее операции целочисленного сложения.
- Что будет с точки зрения простого количества операций? К чему стремится это количество?
- Количество операций умножения каждый раз увеличивается в три раза при увеличении в два раза количества чанков.
- Отношение количества умножений к количеству чанков при стремлении количества чанков к бесконечности тоже стремится к бесконечности.

$$\lim_{k\to\infty} \frac{m_k}{2^k} = \infty. \tag{13}$$

Если наше отношение мы обозначим как $f(k) = \frac{m_k}{2^k}$, то наша задача — найти такую функцию $g(k)$, которая при стремлении k к бесконечности стремится примерно также.

Находим нужную функцию

Такая функция есть и это

$$g(k) = \frac{3^k}{2^k}. \tag{14}$$

Можно сказать, что

$$\lim_{k\to\infty} \frac{f(k)}{g(k)} = 1. \tag{15}$$

Учитываем числа сложений

- Наблюдая над ростом m_k и a_k при росте k, видим закономерность, что их отношение тоже к чему-то стремится (оно стремится к $\frac{1}{6}$, но мы пока не будем это доказывать).
- Для любознательных: это можно доказать и через пределы, и через индукцию, которой мы тоже посвятим немало времени.
- Если нас интересует скорость роста *общего количества операций*, то есть *суммы* $m_k + a_k$, то и тут имеется предел:

$$\lim_{k\to\infty} \frac{m_k + a_k}{g(k)} = c. \tag{16}$$

  Здесь c — константа.
- Этот факт докажет мастер-теорема о рекурсии.

Итоговая сложность

- Итак, функция количества операций в алгоритме Карацубы растёт примерно с такой же скоростью (предел при стремлении аргумента к бесконечности равен единице), как функция $g(k) = \dfrac{3^k}{2^k}$.
- Обычно сводят запись к явным степеням

$$\Theta(k^{\log_2 3}) \approx \Theta(k^{1.58}). \tag{17}$$

- Школьный алгоритм имеет сложность $\Theta(k^2)$. При $k = 1024$ школьный в 18 раз медленнее.
- В реальной жизни, где умножение — достаточно длительная операция, разница будет в сотню раз.

Снова числа Фибоначчи

Вернёмся к нашим числам Фибоначчи и попытаемся решить задачу: найти число Фибоначчи номер k. Вычисляем последовательно? Сложность вычисления будет $\Theta(k)$. Для рекуррентных последовательностей ищут (и часто находят) *производящие функции*.

$$F_k = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^k - \left(\frac{1-\sqrt{5}}{2}\right)^k}{\sqrt{5}}. \tag{18}$$

Интересно, что это за формула. Давайте подставим туда какие-то числа.

Подставляем числа

$$
\begin{aligned}
F_1 &= \frac{\left(\frac{1+\sqrt{5}}{2}\right)^1 - \left(\frac{1-\sqrt{5}}{2}\right)^1}{\sqrt{5}} = \frac{\sqrt{5}}{\sqrt{5}} = 1; \\
F_2 &= \frac{\left(\frac{1+2\sqrt{5}+5}{4}\right) - \left(\frac{1-2\sqrt{5}+5}{4}\right)}{\sqrt{5}} = \frac{\frac{4\sqrt{5}}{4}}{\sqrt{5}} = 1; \\
F_3 &= \frac{\left(\frac{1+3\sqrt{5}+3\cdot 5+5\sqrt{5}}{8}\right) - \left(\frac{1-3\sqrt{5}+3\cdot 5-5\sqrt{5}}{8}\right)}{\sqrt{5}} = \frac{\frac{16\sqrt{5}}{8}}{\sqrt{5}} = 2;
\end{aligned}
\tag{19}
$$

Можно воспользоваться биномиальным разложением. Оставим эту задачу на будущее.

Нашли ли мы решение?

- Мы нашли *математическую* волшебную палочку, которая даёт нам точное *математическое* решение задачи.
- Но наш исполнитель, компьютер, не умеет работать с *точным* представлением вещественных чисел.
- Можно извлечь квадратный корень из 5. Будет ли его квадрат в точности равен пятёрке?

```
1: procedure TEST49
2:     if 1.0 / 49.0 * 49.0 = 1.0 then
3:         Output «All OK!»
4:     else
5:         Output «Something went wrong»
6:     end if
7: end procedure
```

- Борьбе с вещественными числами у нас будет посвящена отдельная тема.
- А пока резюмируем, что точно посчитать числа Фибоначчи, особенно большие, с помощью чистой математики у нас не выйдет.

Позовём на помощь линейную алгебру

Введём вектор-столбец $\begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$, состоящий из двух элементов последовательности Фибоначчи, и умножим его справа на матрицу $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$:

$$\begin{pmatrix} F_0 \\ F_1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} F_1 \\ F_2 \end{pmatrix}. \tag{20}$$

Для вектора-столбца из элементов F_{n-1} и F_n умножение на ту же матрицу даст:

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_{n-1} \\ F_n \end{pmatrix} = \begin{pmatrix} F_n \\ F_{n-1} + F_n \end{pmatrix} = \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}. \tag{21}$$

Таким образом,

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix} = \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}. \tag{22}$$

Быстрое возведение в степень

- Для нахождения n-го числа Фибоначчи достаточно возвести матрицу $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$ в n-ю степень.
- Можно ли возвести число в n-ю степень за число операций, меньших $n - 1$?
- Возведение числа в квадрат есть умножение числа на себя, и достаточно быстро приходит в голову, что, например, возведение в 16-ю степень можно произвести не за 15 операций умножения, а всего за 4:

$$x^{16} = (x^8)^2 = ((x^4)^2)^2 = (((x^2)^2)^2)^2. \tag{23}$$

- С показателями степеней, не равными степеням двойки, вроде бы всё не так просто. Эксперименты над числом 18 покажут нам, что и здесь всё хорошо:

$$x^{18} = (x^9)^2 = (x^8 \cdot x)^2 = (((x^2)^2)^2 \cdot x)^2. \tag{24}$$

- Выведем рекуррентную формулу возведения в степень:

$$x^n = \begin{cases} 1, & \text{если } n = 0 \\ (x^{\frac{n}{2}})^2 & \text{если } n \neq 0 \wedge n \pmod 2 = 0 \\ (x^{n-1}) \times x & \text{если } n \neq 0 \wedge n \pmod 2 \neq 0 \end{cases} \tag{25}$$

- Как оценить сложность этого алгоритма?
- Представим степень, в которую мы возводим, в виде двоичного числа, например, степень 25 в виде 11001.
- Тогда нечётная степень будет означать, что последний разряд в двоичном представлении степени есть единица и операция $n - 1$ есть её обнуление.
- Чётная же степень будет означать, что последний разряд равен нулю и деление такого числа на два есть вычёркивание этого разряда.
- Каждую из единиц требуется уничтожить, не изменяя количества разрядов, и каждый из разрядов требуется уничтожить, не изменяя количества единиц.
- Таким образом, сложность алгоритма равна $\Theta(\log N)$.
- Алгоритмы работы с битами слова основаны на законах математической логики и мы этим вскоре займёмся.