АЛГОРИТМЫ И МОДЕЛИ ВЫЧИСЛЕНИЯ

Вас ждут две (или три) курсовые контрольные. Они будут проходить, соответственно, 20.03 (midterm), 15.05 (final) и 22.05 (утешительная) в Б.Хим. и Акт.зале с 13:55 – 17:05.

Обязанности сторон и правила аттестации (протокол)

1. В нашем курсе поддерживается режим свободного перехода. Обязательным условием перехода является **согласие принимающей стороны**. Для оформления перехода нужно написать заявление, подписать у обоих семинаристов и отнести его на кафедру. Процесс должен завершится в течение трех недель.

2. Протокол получения оценки близок к протоколу, использованному на ТРЯП и состоит из двух пунктов.

(*i*) Студент, получивший положительную оценку по обеим курсовым контрольным и (условно) "сдавший задание"— это означает, что семинарист положительно оценивает вашу работу в семестре (у каждого семинариста могут быть свои критерии, см. ниже) — имеет право получить зачетную оценку, равную max(полусумма курсовых контрольных, оценка семинариста).

Для этой категории студентов повышение оценки выше протокольной возможно, если это допускает ваш семинарист. Если ваш семинарист считает, что ваша оценка заслуженная, а вы не согласны с этим, то этот вопрос может быть рассмотрен в присутствии всех заинтересованных сторон комиссией.

(*ii*) Студент, получивший хотя бы одну неудовлетворительную оценку на курсовых контрольных или пропустивший одну из них (**даже по уважительной причине**), обязан писать утешительную контрольную, которая может быть дополнена устным опросом.

(*ii*) -а. Если утешительная контрольная написана на положительную оценку, то вопрос о зачете передается семинаристу. И также, если вы настаиваете, что ваша работа недооценена (или переоценена), то этот вопрос может быть рассмотрен в присутствии всех заинтересованных сторон комиссией.

(*ii*)-б. Если утешительная контрольная не написана на положительную оценку, то для получения зачетной оценки нужно следить за графиком пересдач (и процесс продолжается).

3. Идущее ниже задание является **официальным**. В частности, это означает, что при возникновении каких-то коллизий (например, при незачетах и/или стремлении повысить оценку, полученную по протоколу и т.д.) лично я буду исходить из предположения, что студенты представляют себе, как решать задачи из этого задания.

Каждый семинарист имеет право вносить в него необходимые изменения. Кроме того, в конце курса семинаристы будут оценивать работу студентов. Эта оценка может весьма существенно сказаться на зачетной оценке.

ПРОГРАММА КУРСА

Приводим понедельный план наших занятий.

В течение *первых двух недель* мы повторим формальное определение алгоритма и связанные понятия. Параллельно будут разобраны различные алгоритмы. При этом основное внимание будет уделяться проверке корректности конкретных процедур и оценке их сложности. В связи с последним мы разберем некоторые стандартные приемы получения асимптотических оценок.

1. **Темы 1-й недели 6.02-12.02. 1 лекция.**

Формальное определение алгоритма. Емкостная и времéнная сложность алгоритма. Различные определения сложности алгоритма. Эффективные (полиномиальные алгоритмы). Примеры алгоритмов и иллюстрация принципов разработки алгоритмов: проверка простоты и факторизация чисел; сортировка слиянием; Линейный алгоритм поиска медианы.

Асимптотические оценки. Нотация: $O(\cdot)$, $o(\cdot)$, $\Omega(\cdot)$, $\omega(\cdot)$, $\Theta(\cdot)$.

2. **Темы 2-й недели 13.02-19.02. 2-я лекция.**

Модели вычислений. Формальное определение алгоритма. Разрешимые и перечислимые языки[1] Теорема о линейном ускорении. Теоремы о времéнной и емкостной иерархии (без доказательства). Определения сложностных классов ($\mathcal{P}, \mathcal{NP}, \mathcal{PSPACE}$).

Примеры алгоритмов и иллюстрация принципов разработки алгоритмов: алгоритм Евклида; индийское возведение в степень; одновременное вычисление максимального и минимального элементов в массиве; поиск ближайшей пары точек; быстрое умножение чисел и матриц: (алгоритмы Карацубы и Штрассена).

Асимптотические оценки. Нотация: $O(\cdot)$, $o(\cdot)$, $\Omega(\cdot)$, $\omega(\cdot)$, $\Theta(\cdot)$. Основная теорема о рекуррентных оценках (нахождение асимптотики рекуррентности вида $T(n) = aT(\frac{n}{b}) + f(n)$). Дерево рекурсии. Линейный вероятностный алгоритм нахождения медианы. Решение линейных рекуррентных уравнений.

Следующие четыре лекции будут посвящены теории сложности вычислений, а более конкретно, теории NP-полноты и связанным понятиям. По окончании будет проведен промежуточный курсовой тест в понедельник 20.03 с 13:55 по 17:05 в Актовом зале и Б. Хим).

Кроме того, лекция в субботу 25.02 будет перенесена на понедельник 27.02 10:45–12:10 в 110КПМ. Семинар в группе 577 будет перенесен на понедельник 12:20–13:55 в 432 ГК.

4. **Темы 3–4 недель 20.02-5.03. 3-я лекция пройдет в понедельник 27.02 10:45–12:10 в 110КПМ**

Класс $\mathcal{P}$. Примеры языков из $\mathcal{P}$: принадлежность слова регулярному языку; принадлежность слова КС-языку; системы линейных уравнений (полиномиальная реализация метода Гаусса). Классы NP и co-NP. Примеры языков из NP: выполнимость, простые числа; пересечение регулярных языков, заданных конечными автоматами; непланарные графы.

5. **Темы 4–5 недель 27.02-1.03 4–5 лекции.**

[1] Повторение того, что вы изучали в курсе ТФС три месяца назад. Исходя из стабильного прошлого опыта, для многих эта тема окажется совсем не повторением, а, если так уместно выразиться, откровением.

Полиномиальная сводимость. Сводимость по Карпу и по Куку (по Тьюрингу). Теорема Кука-Левина. Примеры полиномиально полных языков: выполнимость; протыкающее множество; максимальное 2-сочетание; 3-сочетание; вершинное покрытие; клика; хроматическое число; гамильтонов цикл; рюкзак; разбиение; максимальный разрез; N[ot]A[ll]E[qual]-выполнимость.

Класс $\mathcal{PSPACE}$. Теорема Савича (без доказательства). Полные языки в $\mathcal{PSPACE}$: истинность булевской формулы с кванторами; эквивалентность конечных автоматов.

6. **Темы 6-й недели 13.03-20.03. 6-я лекция.**

Теория NP-полноты.

Числовые алгоритмы[2]. Обобщенный алгоритм Евклида. Решение линейных диофантовых уравнений. Модульная арифметика. Китайская теорема об остатках. Функция Эйлера. Первообразные корни. Кольца Z_n, в которых существуют первообразные корни. Индексы (дискретные логарифмы). Кодирование с открытым ключом. Квадратичные вычеты. Схемы RSA шифрования и цифровой подписи, дискретное логарифмирование. Протокол Диффи-Хелмана.

В понедельник 20.03 с 13:55–17:05 в Актовом зале и Большой химической пройдет первая курсовая контрольная работа по темам 1–6. Явка на нее обязательна.

По протоколу студенты, получившие неудовлетворительную оценку на первой или второй контрольной, не могут быть аттестованы без получения положительной оценки на последующих контрольных (как это происходит на ТРЯП). К этой группе относятся и студенты, пропустившие контрольную даже по уважительной причине.

7. **Темы 7-й недели 20.03-26.03. 7 лекция.**

Структуры данных. Стек и очередь. Двоичная куча (пирамида). Двоичное дерево поиска и его сбалансированные вариации. Амортизационный анализ. Структура "union-find" для хранения системы непересекающихся множеств. Хеш-таблицы. Разрешение коллизий с помощью цепочек. Хеш-функции (деление с остатком, умножение). Универсальные и k-универсальные хеш-функции.

8. **Темы 8-й недели 27.03-2.04 8-я лекция.**

Дискретное преобразование Фурье (ДПФ); алгоритм быстрого преобразования Фурье (БПФ); перемножение многочленов с помощью БПФ. Поиск подстрок. Использование БПФ для распознавания образцов. Циркулянты. Решение линейных уравнений с циркулянтными матрицами.

9. **Темы 9-й недели 3.04-9.04. 9-я лекция.**

Алгоритмы сортировки: "пузырек, "быстрая сортировка (quicksort); сортировка с помощью кучи (heapsort); сортировка слиянием (mergesort). Анализ трудоемкости алгоритма quicksort по наихудшему случаю и в среднем. Устойчивость алгоритма сортировки. Цифровая сортировка. Порядковые статистики. Нижние оценки в модели разрешающих деревьев.

10. **Темы 10-й недели и 11-й недели 10.04-23.04. 10–11-я лекции.**

Потоки и разрезы в сети. Теорема о максимальном потоке и минимальном разрезе. Понятие остаточного графа и увеличивающего пути. Метод Форда-Фалкерсона для вычисления максимального потока и минимального разреза. Обобщения потоковой сети (пропускные способности узлов и пр.). задача о максимальном паросочетании в двудольном графе. Задача линейного программирования. Основные понятия. Выпуклые многогранники. Теорема двойственности. Задача назначении.

11. **Темы 11-й – 12-й недель 17.04-30.04. 11–12-я лекции.**

Алгоритмы на графах: поиск в ширину; поиск в глубину; определение двусвязных и/или сильносвязных компонент; топологическая сортировка. Остовные деревья: алгоритмы Прима и Краскала. Кратчайшие пути: алгоритмы Дейкстры, Флойда, Беллмана–Форда. Паросочетания.

12. **Тема 13-й недели 1.05-7.05. 13-я лекции.**

Вероятностные алгоритмы. Классы $\mathcal{RP}$, $\mathcal{BPP}$, $\mathcal{ZPP}$. Вероятностные алгоритмы: проверка простоты; вычисление медианы массива; проверка полиномиальных тождеств; поиск паросочетаний в графах; алгоритм Каргера поиска минимального разреза. Приближённые вероятностные алгоритмы поиска максимального разреза. Лемма Шварца–Зиппеля. Дерандомизация.

14. **Тема 14-й недели 8.05-14.05. 14-я лекция.**

Методы решения переборных задач: динамическое программирование, шкалирование, ветви и границы, приближенные алгоритмы для задачи максимального разреза. ε-оптимальная процедура решения задачи о рюкзаке.

Заключительная контрольная понедельник 15.05 с 13:55–17:05 в Акт. зале и Б.Хим.

Утешительная контрольная в понедельник 22.05 с 13:55–17:05 в Акт. зале и Б.Хим .

ЛИТЕРАТУРА

Основная

1. **[АХУ]** *Ахо А., Хопкрофт Д., Ульман Д.* Построение и Анализ Вычислительных Алгоритмов. М.: Мир, 1979.
2. **[ГД]** *Гери М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982.
3. **[ДПВ]** *Дасгупта С., Пападимитриу Х., Вазирани У.* Алгоритмы. М.: МЦНМО, 2014.
4. **[Кормен 1]** *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: Построение и Анализ. М.: МЦНМО, 2002.
5. **[Кормен 2]** *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы: Построение и Анализ. (2-е изд.) М.: Вильямс, 2005.
6. **[Кормен 3]** *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы: Построение и Анализ. (3-е изд.) М.: Вильямс, 2013.

[2]Формально повторяются темы второго семестра (за возможным исключением алгоритма RSA.

7. **[ХМУ]** *Хопкрофт Д., Мотвани Р., Ульман Д.* Введение в теорию автоматов, языков и вычислений. М.,: Вильямс, 2002.
8. **[AB]** *Arora S., Barak B.* Computational Complexity: A Modern Approach. theory.cs.princeton.edu/complexity/book.pdf
9. **[GL]** *Gács P., Lovasz L.* Complexity of Algorithms. www.cs.elte.hu/∼lovasz/complexity.pdf

Дополнительная

1. *Верещагин Н., Шень А.* Вычислимые Функции. М.: МЦНМО, 1999. (Электронный вариант: www.mccme.ru/free-books)
2. **[Виноградов]** *Виноградов И.* Основы теории чисел. М.-Л.: Гостехиздат, 1952
3. *Вялый М., Журавлев Ю., Флеров Ю.* Дискретный анализ. Основы высшей алгебры. М.: МЗ Пресс, 2007.
4. **[К-Ш-В]** *Китаев А., Шень А., Вялый М.* Классические и квантовые вычисления. М.: МНЦМО-ЧеРо, 1999.
5. **[Кнут-1, Кнут-2, Кнут-3, Кнут-4] (цифра отвечает номеру тома** *Кнут Д.* Искусство программирования для ЭВМ. Существует несколько изданий на русском. Первое было выпущено издательством "Мир" в семидесятых. В настоящее время выпускается издательством "Вильямс". Есть многочисленные сетевые варианты.
6. **[К-Ф]** *Кузюрин Н., Фомин С.* Эффективные алгоритмы и сложность вычислений. М.: МФТИ, 2007.
7. **[П-С]** *Пападимитриу Х., Стайглитц К.* Комбинаторная оптимизация. Алгоритмы исложность. М.: Мир, 1985.
8. **[Хинчин]** *Хинчин А.* Цепные дроби. М.: Наука, 1979.
9. **[Ш]** *Шень А.* Программирование. Теоремы и задачи. М.: МЦНМО, 2007. (Электронный вариант: www.mccme.ru/free-books)
10. *Lovasz L.* Computational complexity. www.cs.elte.hu/∼lovasz/complexity.pdf

Задание на первые шесть недель

Задание состоит из списка недельных заданий (указаны даты обсуждения на семинарах и/или сдачи соответствующих задач). Каждое недельное задание состоит из четырех-пяти обязательных задач и одной-двух дополнительных задач (дополнительные задачи выделены буквой «Д»). Решение дополнительных задач не обязательно, но может быть полезно студентам, претендующим на более высокую оценку.

Обозначения

Пусть $f(n)$ и $g(n)$ — неотрицательные функции.

- $f(n) = O(g(n))$ означает, что порядок роста g при $n \to \infty$ *не меньше* порядка роста f, т. е. $\exists c > 0|$ при $n > n_0\, f(n) \le cg(n)$;
- $f(n) = \Omega(g(n)) \Leftrightarrow g(n) = O(f(n))$ (порядок роста f не меньше порядка роста g);
- $f(n) = (g(n))$, если $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ (f имеет меньший порядок роста, чем g);
- $f(n) = \omega(g(n)) \Leftrightarrow g(n) = o(f(n))$ (g имеет меньший порядок роста, чем f);
- $f(n) = \Theta(g(n))$, если $f(n) = O(g(n))$ и $f(n) = \Omega(g(n))$ (порядки роста f и g одинаковы).

По определению, $\log^* M = \min\{k \mid \underbrace{\log\log\ldots\log M}_{k \text{ раз}} \le 1\}$.

$\lfloor a \rfloor$ обозначает наибольшее целое, не превосходящее число a
$\lceil a \rceil$ обозначает наименьшее целое, превосходящее a

Необходимо знать какой-нибудь вывод формулы Стирлинга $n! \sim \sqrt{2\pi n}(\frac{n}{e})^n$ и суммы гармонического ряда $\sum_{i=1} n\frac{1}{i} = \ln n + \gamma$, где $gamma = 0.57721\ldots$ — это так называемая константа Эйлера. Кроме того, нужно знать, что такое числа Каталана $c_n = \frac{1}{n+1}\binom{2n}{n}$ (c_n имеет множество комбинаторных интерпретаций, например, равно правильно построенных скобочных выражений или путей Дика длины $2n$) и выражение для их производящей функции $D(t) \stackrel{def}{=} \sum_{n=0}^{\infty} c_n x^n = \frac{1-\sqrt{1-4x}}{2x}$.

Задание на первую неделю: 06.02–12.02 [0.1]
Примеры алгоритмов. Оценки
Раздел 1–2 программы
Литература: [Кормен 1, Введение. Глава 1] [Кормен 3, Главы 1–3], [ДПВ. Гл. 1–2]

Информация для моих студентов. После задания в квадратных скобках стоит проходной балл. После каждой задачи – ее стоимость.

Задача 1. (0.04). Дан массив из n элементов, на которых определено отношение равенства (например, речь может идти о массиве картинок или музыкальных записей). Постройте алгоритм, который в "потоковом режиме обработки данных"[3] определяет, есть ли в массиве элемент, повторяющийся больше $\frac{n}{2}$ раз. Считается, что в вашем распоряжении есть память объемом $O(\log n)$ битов.

Замечание. Мы еще вернемся к этой задаче. Эта сентенция здесь и далее означает, что тематика этой задачи любопытна и будет выделена в отдельный сюжет (но, возможно, он будет изложен в задачах с пометкой **"Д"** *из-за недостатка времени).*

Задача 2. (2×0.01). Дано описание программы.

- ВХОД(x, y– натуральные числа).
- Пусть 2^{d_x}– максимальная степень 2, делящая x; d_y– определяется аналогично.
- Положим $a = x2^{-d_x}$, $b = y2^{-d_y}$.
- Поменять местами b и a, если $b < a$.
- ПОКА $b > 1$ ВЫПОЛНИТЬ
- Положим r равным тому из чисел $a+b$, $a-b$, которое делится на 4.
- Положим $a = \max(b, r2^{-d_r})$, $b = \min(b, r2^{-d_r})$, где 2^{d_r}– максимальная степень 2, делящая r.
- КОНЕЦ ЦИКЛА ПОКА
- ВЫХОД($a2^{\min(d_x,d_y)}$).

[3] Поточный алгоритм (англ. streaming algorithm или on-line algorithm) — алгоритм для обработки последовательности данных в один или малое число проходов. В этой задаче предусматривается ровно два прохода.

Заметим, что рассматриваются и еще более жесткие дисциплины, например, алгоритмы "в реальном времени", когда результат начинает печататься с константной задержкой, которая не зависит от длины входа. Например, так фактически изначально строились схемы умножения битовых чисел (старшие биты выдавались до того, как были считаны оба операнда). Эта идеология оказала существенное влияние на архитектуру т. н. суперкомпьютеров, причем, как это часто бывает в истории, некоторые выдающиеся персонажи, например, Seymour Cray, обязательно посмотрите, кто это такой, сейчас безвременно забыты, хотя их вклад в информатику вообще не поддается никакому измерению.

(i) Программа, если в ней исправить неточности, вычисляет известную функцию. Что эта за функция?

(ii) Оцените трудоемкость (скорректированной) процедуры (число производимых битовых операций), если x, y — n-битовые чмсла.

Задача 3. (0.02). Нужно отсортировать массив n чисел, размер которого превышает размер оперативной памяти компьютера в k раз. Предложите как можно более быструю процедуру и оцените ее трудоемкость.

Замечание 1. Адаптируйте для этой задачи сортировку слиянием с учетом вспомогательной памяти, который требует этот алгоритм.

Замечание 2. Нужно записать отсортированный массив во внешнюю память, используя как можно меньше попарных сравнений, которые можно проводить только в (быстрой) оперативной памяти.

Мы еще вернемся к этой задаче и попробуем оценить ее трудоемкость снизу.

На самом деле, даже небольшие и вполне оправданные с точки зрения конкретных практических приложений уточнения этой задачи, например, формализация обращения к быстрой и медленной памяти, может потребовать значительно больше усилий, чем кажется на первый взгляд. Если кого-то этот вопрос заинтересует, то можно порекомедовать ознакомиться с главой 5 третьего тома монографии Д. Кнута **[Кнут-3]**.

Задача 4. (3×0.02). На вход подается описания n событий в формате (s, f) — время начала и время окончания. Требуется составить расписание для человека, который хочет принять участие в максимальном количестве событий. Например, события это доклады на конфереции или киносеансы на фестивале, которые проходят в разных аудиториях. Предположим, что участвовать можно только с начала события и до конца. Рассмотрим три жадных алгоритма.

1. Выберем событие кратчайшей длительности, добавим его в расписание, исключим из рассмотрения события, пересекающиеся с выбранным. Продолжим делать то же самое далее.

2. Выберем событие, наступающее раньше всех, добавим его в расписание, исключим из рассмотрения события, пересекающиеся с выбранным. Продолжим делать то же самое далее.

3. Выберем событие, завершающееся раньше всех, добавим его в расписание, исключим из рассмотрения события, пересекающиеся с выбранным. Продолжим делать то же самое далее.

Какой алгоритм вы выберете? В качестве обоснования для каждой процедуры проверьте, что она является оптимальной (т. е. гарантирует участие в максимальном числе событий) или постройте конкретный контрпример.

Задача 5. (0.01). Найдите Θ-асимптотику числа BR_{4n+2} правильных скобочных выражений длины $4n+2$.

Замечание: это простая задача для проверки, как вы умеете обращаться с формулой Стирлинга.

Задача Д–1. (0.02). Найдите явное аналитическое выражение для производящей функции чисел BR_{4n+2} (ответ в виде суммы бесконечного ряда не принимается).

Замечание: Мы еще вернемся к этой задаче и попробуем придумать общий метод решения задач такого типа.

Задание на вторую неделю: 13.02–19.02 [0.1]
Оценки. Рекуррентные последовательности. Раздел 3 программы
Литература: [Кормен 1, Глава 1] [Кормен 3, Главы 3–5], [ДПВ. Гл. 2]

Задача 6. ($0.02 + 0.03$).

Дана рекурсивная программа

```
функция f(n)
  if n > 1
     печать("алгоритм")
     печать("алгоритм")
     печать("алгоритм")
     f(⌊n/2⌋)
     f(⌊n/4⌋)
  endif
```

Пусть $g(n)$ обозначает число слов "алгоритм", которые напечатает программа.

(i) Найдите в виде функции от n Θ-асимптотику $g(n)$.
(ii) Считая n степенью двойки, вычислите $g(n)$ точно.

Задача 7. (0.02).

Оцените трудоемкость рекурсивного алгоритма, разбивающего исходную задачу размера n на три задачи размеров $\lceil \frac{n}{\sqrt{3}} \rceil - 5$, используя для этого $10\frac{n^3}{\log n}$ операций.

Задача 8. (0.02). Рассмотрим детерминированный алгоритм поиска медианы по кальке известного линейного алгоритма, где используется разбиение массива на четвёрки элементов, в каждой из которых определяется *нижняя* медиана, т. е. из в каждой четверки выбирается второй по порядку элемент (элементы можно считать различными). Приведите рекуррентную оценку числа сравнений в этой процедуре и оцените сложность такой модификации.

Задача Д–2. (0.02). Оцените трудоемкость рекурсивного алгоритма, разбивающего исходную задачу размера n на n задач размеров $\lceil \frac{n}{2} \rceil$ каждая, используя для этого $O(n)$ операций.

Задача Д–3. (0.03). Функция натурального аргумента $S(n)$ задана рекурсией:

$$S(n) = \begin{cases} 100 & n \le 100 \\ S(n-1) + S(n-3) & n > 100 \end{cases}$$

Оцените число рекурсивных вызовов процедуры $S(\cdot)$ при вычислении $S(10^{12})$.

Задача Д–4. (0.02). Оцените как можно точнее высоту дерева рекурсии для рекуррентности $T(n) = T(n - \lfloor\sqrt{n}\rfloor) + T(\lfloor\sqrt{n}\rfloor) + \Theta(n)$.

Полиномиальные алгоритмы
Раздел 4 программы
Литература: [Кормен 1, Глава 36]
[Кормен 3, Глав 34], [ДПВ. Глава 8]

Немного теории: класс $\mathcal{P}$

Мы приступаем к исследованию формальных свойств алгоритмов, трудоемкость которых полиномиальна по длине входа. Напомню, что такие алгоритмы мы считаем эффективными[4].

Ниже идет краткое теоретическое введение по разделу 3 программы, хотя формально в это задание включены только задачи о классе $\mathcal{P}$.

Пусть фиксирован алфавит Σ (если специально не оговорено, то будем считать, что $\Sigma = \{0, 1, *(\text{разделитель})\}$). Вспомним, что *предикат*— это булева функция на словах $P(\cdot) : \Sigma^* \to \{0, 1\}$, и любому предикату можно поставить в соответствие язык всех слов, на которых он истинен: $\{x \in \Sigma^* | P(x) = 1)\}$. Класс $\mathcal{P}$ состоит из всех *полиномиально вычислимых предикатов* или *языков*, которые распознаются *полиномиальными алгоритмами*. Иными словами, любой предикат $P(\cdot) \in \mathcal{P}$ вычисляется на произвольном входе x за время $poly(|x|)$, где $|x|$— длина слова x или длина кодировки входа x. А любому полиномиальному алгоритму T — вычислимой функции, перерабатывающей *слова-входы* x_i в слова-выходы *ответы* y_i, $i = 1, 2, \dots$ можно сопоставить ее *график* полиномиальный предикат: $L_T = \{x_1 * y_1, x_2 * y_2, \dots\} \in \mathcal{P}$.

На любой универсальный язык программирования (иначе говоря, на любую универсальную МТ) можно наложить естественные *синтаксические* ограничения, выделяющие полиномиальные алгоритмы. Например, если использовать "Паскаль", то нужно отказаться от использования GOTO, REPEAT и WHILE, а все циклы FOR должны иметь полиномиальную по длине входа границу. Наоборот, любой полиномиальный алгоритм может быть оформлен с приведенными синтаксическими ограничениями.

Следующее замечание-вопрос носит полуфилософский характер: а почему используются именно полиномы? Ответ (опять же полуфилософский): прежде всего, поскольку они замкнуты относительно суперпозиции, поэтому, если программа, выполняющаяся за полиномиальное по входу время, будет фиксированное число раз вызывать любые подпрограммы, также выполняющиеся за полиномиальное время, то и результирующая программа также будет выполняться за полиномиальное время. **Контрольный вопрос: сохранится ли ответ, если допустить, скажем, линейное по входу число обращений к (полиномиальным) подпрограммам?**

Следующий шаг, который мы сделаем, в каком-то смысле прямо противоположен деятельности по разработке и анализу эффективных алгоритмов [для конкретных задач]. Мы начнем изучать методы, позволяющие заключить, что подобных алгоритмов [для этих конкретных задач] не существует. Мы начали обсуждать эту проблему на занятиях, когда, например, коснулись (информационной) нижней оценки $\Omega(n \log n)$ сортировки попарными сравнениями. **Эта тема является одной из центральных тем курса. Она связана с описанием классов $\mathcal{P}, \mathcal{NP}$ и $co-\mathcal{NP}$ и понятием полиномиальной сводимости.** Трудности с изучением этой темы аналогичны тем, которые возникли, например, в прошлом году при изучениии регулярных языков: отсутствие в предыдущем опыте каких-то формальных или неформальных картинок, помогающих правильно сориентироваться. Это, прежде всего, означает, что стандартный метод обучения: начать изучение за три минуты до экзамена или зачета, может оказаться крайне непродуктивным. К этой теме нужно просто "привыкнуть", что подразумевает постоянное продумывание возникающих вопросов.

[4]Хотя каждый понимает, что это некоторое теоретическое преувеличение. Как, например, использовать алгоритм, временная сложность которого оценивается полиномом степени гугл? На самом деле, неформальный смысл дискриминации полиномиальный — неполиномиальный гораздо глубже. Оказывается, дальше идет чистый лозунг — или лучше сказать — тезис, что попытки уточнения уже известных оценок и/или попытки построить полиномиальные алгоритмы приводят к существенному расширению нашего понимания мира алгоритмов вообще. А последний, как мы уже понимаем, несмотря на внешнюю простоту описания, может быть весьма хитрым и контраинтуитивным.

Задача 9. (3×0.01). Докажите , что следующие языки принадлежат классу $\mathcal{P}$. Можно считать, что графы кодируются соответствующими матрицами смежности.

(*i*) Язык двудольных графов, содержащих не менее 2017 треугольников (трех попарно смежных вершин).
(*ii*) Язык несвязных графов без циклов.
(*iii*) Язык квадратных $\{0, 1\}$-матриц порядка $n \geq 3000$, в которых есть квадратная подматрица порядка $n - 2017$, заполненная одними единицами.

Задача Д–5. (0.02) На клетчатой бумаге закрасили несколько клеток (координаты закрашенных клеток передаются на вход). Лежит ли в классе $\mathcal{P}$ язык, состоящих из описаний таких множеств закрашенных клеток, которые можно замостить доминошками размера 1×2 ровно в два слоя? (Накрыты должны быть только закрашенные клетки и ровно двумя слоями домино.)

Для доказательства принадлежности языка классу $\mathcal{P}$ нужно предъявить и обосновать полиномиальный алгоритм. Для обоснования противоположного утверждения нужно привести убедительные, с вашей точки зрения, аргументы.

Быстрое умножение чисел и матриц
Литература: [Кормен 1, §31.2]
Литература: [Кормен 3, §4.2], [ДПВ, §2.5]

Школьный способ "умножения в столбик" n-битовых чисел заключается в последовательном сложении n двоичных чисел длины $O(n)$, т.е. сложность способа $O(n^2)$. Можно ли умножать числа быстрее?

Рассмотрим умножение n-значных чисел A и B. Представляя A и B в виде $A = A_1 + 2^{\frac{n}{2}} A_2, \quad B = B_1 + 2^{\frac{n}{2}} B_2$, где A_1, A_2, B_1, B_2 — $\frac{n}{2}$-значные числа, находим: $AB = A_1 B_1 + 2^{\frac{n}{2}}((A_1 + A_2)(B_1 + B_2) - (A_1 B_1 + A_2 B_2)) + 2^n A_2 B_2$.

Если не учитывать, что в "среднем" члене разрядность может увеличиться на 1, то получаем рекурсию для числа операций $\varphi(n)$ в алгоритме: $\varphi(n) = 3\varphi(\frac{n}{2}) + O(n)$, откуда по ОТ $\varphi(n) = O(n^{\log 23} = O(n^{1.5849\dots})$.

Следующий трюк позволяет "избавиться" от лишнего разряда. Для этого запишем $A_1 + A_2 = a_1 2^{\frac{n}{2}} + a_2$ и $B_1 + B_2 = b_1 2^{\frac{n}{2}} + b_2$, где a_1, b_1 — биты. Тогда $(A_1 + A_2)(B_1 + B_2) = a_1 b_1 2^n + (a_1 b_2 + a_2 b_1) 2^{\frac{n}{2}} + a_2 b_2$. Член $a_2 b_2$ вычисляется рекурсивно, а остальные вычисляются в линейное время.

Пример работы алгоритма Карацубы (стрелка после произведения указывает на вспомогательные произведения, которые требуется вычислить; рекурсия останавливается на двузначных числах):

1) $216_{10} \times 139_{10} = 11011001 \times 10001011 \to 1101 \times 1000, 1001 \times 1011, (1101 + 1001) \times (1000 + 1011) = 10110 \times 10011$;

2) $1101 \times 1000 \to 11 \times 10 = 110; 01 \times 00 = 0; (11 + 01) \times (10 + 00) = 100 \times 10 = 1000$;

3) $1101 \times 1000 = 1100000 + (1000 - 110 - 0) \times 100 + 0 = 1101000$

4) $1001 \times 1011 \to 10 \times 10 = 100, 01 \times 11 = 11, (10 + 01) \times (10 + 11) = 11 \times 101 = 1111$;

5) $1001 \times 1011 = 1000000 + (1111 - 100 - 11) \times 100 + 11 = 1100011$

6) $010110 \times 010011 \to 010 \times 010 = 100, 110 \times 011, (010 + 110) \times (010 + 011) = 1000 \times 0101 = 101000$

7) $0110 \times 0011 \to 01 \times 00 = 0, 10 \times 11 = 110, (1 + 10) \times (0 + 11) = 11 \times 11 = 1001$;

8) $0110 \times 0011 = 0 + (1001 - 0 - 110) \times 100 + 110 = 10010$

9) $010110 \times 010011 = 100000000 + (101000 - 10010 - 100) \times 1000 + 10010 = 110100010$

10) $11011001 \times 10001011 = 110100000000000 + (110100010 - 1101000 - 1100011) * 10000 + 1100011 = 111010111010011 = 30163_{10}$

Алгоритм Штрассена

Обычный способ перемножения 2×2 матриц требует 8 умножений и 4 сложения:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

В 1969 году Ф.Штрассен (V.Strassen) открыл, что следующую процедуру

$$\begin{array}{lllllll} p_1 & = & a(f-h) & & p_2 & = & (a+b)h \\ p_3 & = & (c+d)e & & p_4 & = & d(g-e) \\ p_5 & = & (a+d)(e+h) & & p_6 & = & (b-d)(g+h) \\ p_7 & = & (a-c)(e+f). & & & & \end{array}$$

Теперь

$$\begin{array}{lllllll} ae+bg & = & p_5+p_4-p_2+p_6 & & af+bh & = & p_1+p_2 \\ cf+dh & = & p_5+p_1-p_3-p_7 & & ce+dg & = & p_3+p_4. \end{array}$$

Таким образом, нам требуется 7 умножений и 18 сложений. На первый взгляд, мы поменяли шило на мыло, но (тут начинается маленькое чудо), обратим внимание на то, что формулы Штрассена справедливы и тогда, когда переменные некоммутативные, а **поэтому возможно рекурсивное применение процедуры для перемножения матриц**! Теперь вспомним, что сложение двух $n \times n$ матриц требует только $O(n^2)$ операций. Поэтому, если рекурсивно запустить алгоритм (подробности можно прочитать, например в **[Кормен 1, §31.2]**), т.е. разбить матрицу порядка $n \times n$ на четыре блока порядка $\frac{n}{2} \times \frac{n}{2}$ и провести перемножения блокам по формулам, записанным выше[5], то получится такая рекуррентная оценка трудоемкости алгоритма перемножения матриц: $T(n) = 7T(\frac{n}{2}) + O(n^2)$.

Попробуйте запрограммировать алгоритм Штрассена (рекурсия должна обрываться на матрицах небольшого порядка) и *явно* оцените порядок матриц, для которых *полное* число операций алгоритма Штрассена становится *меньше*, чем у классического[6]

Это дополнительный материал, но в программу входит только *быстрое возведение в степень*

"Индийское" возведение в степень

Аддитивные цепочки

Литература: Д.Кнут. Искусство программирования для ЭВМ, М.: Мир, 1977, т.2 §4.6.3 "Вычисление степеней".

Попробуем проиллюстрировать цели нашего курса, а также возникающие при этом трудности, на примере простой задачи.

ПРОБЛЕМА: даны натуральные числа a и n в двоичной записи. Нужно построить алгоритм для вычисления a^n.

Вообще говоря, решение этой проблемы известно каждому с первого класса (а возможно, и раньше), поскольку, если не вдаваться "в детали", то речь идет о последовательном $n-1$-кратном умножении на a. Итак, искомый алгоритм фактически строится тривиально, причем по очевидным причинам он корректный.

Но, как известно, дьявол кроется в деталях, и такая процедура нас не устраивает по следующей причине: для ее выполнения нужно порядка n умножений. Оценим сложность алгоритма. Представим себе, что умножение занимает один такт времени[7]. Тогда алгоритм требует $n-1$ тактов. Скажем, если двоичная длина n порядка 1000 бит, то алгоритм будет работать 2^{1000}, а это число значительно превышает число протонов во Вселенной.

Зафиксируем этот факт. Нам удалось построить алгоритм решения задачи, и даже проанализировать его корректность. Это уже большое достижение. Но нас не удовлетворяет трудоемкость процедуры, и мы естественно переходим к вопросу, нельзя ли предложить более быстрый алгоритм для этой задачи.

Способ исправления ситуации многим, я уверен, известен. Его подсказывает известный способ вычисления степеней двойки: вместо последовательного умножения будем последовательно возводить промежуточные результаты в квадрат: $x_0 \leftarrow a, x_1 \leftarrow x_0^2, \ldots, x_k \leftarrow x_{k-1}^2$. Ясно, что $x_k = a^{2^k}$. Использование этой идеи приводит к следующему алгоритму возведения в степень: нужно последовательно сканировать биты показателя n справа налево и в зависимости от четности просматриваемого бита умножать текущий результат на a или возводить в квадрат (программа ниже взята из книги А.Шеня "Программирование: теоремы и задачи"):

```
k := n; b := 1; c:=a;
{a в степени n = b * (c в степени k)}
while k <> 0 do begin
| if k mod 2 = 0 then begin
| | k:= k div 2;
| | c:= c*c;
| end else begin
| | k := k - 1;
| | b := b * c;
| end;
end;
```

Оказывается, что в старинном "индийском алгоритме" (другое название *бинарный алгоритм*) используется похожая идея, но биты показателя сканируются слева налево (просмотр идет от старших битов к младшим).

Описание алгоритма следующее. В двоичной записи показателя n (нули слева убраны) произведем следующую замену символов $1 \to SA$ и $0 \to S$. В полученном слове удалим начальную пару символов SA. В результате получим "код программы" на следущем диалекте: S нужно понимать как возведение в квадрат текущего результата, а A как умножение на a. Тогда, начиная с a и двигаясь по коду слева направо, в конце вычисления мы получим a^n.

Пусть $\nu(n)$ — число единиц в двоичной записи n, а $\lambda(n) = \lfloor \log_2 n \rfloor$. Тогда в индийском алгоритме будет проведено $\lambda(n) + \nu(n) - 1$ умножений (а в первом рассмотренном алгоритме будет на одно умножение больше).

Зафиксируем этот факт. Мы построили целых два алгоритма и даже можем попытаться доказать их корректность. Действовать можно по индукции, и фактически для первой процедуры индукционная гипотеза записана в качестве комментария во второй строке программы. Для индийского алгоритма рассуждения аналогичны. такая ситуация, конечно, необычная, поскольку стандартная практика не предполагает проверки корректности, и вовсе не потому, что и так "все ясно", а из практической невозможности провести анализ для сколько-нибудь нетривиальной процедуры. Поэтому часть просто не понимает необходимость проверки, а (меньшая) часть видит, что это очень сложно, и ограничивается ритуальными восклицаниями: "...очевидно.. и пр.". Можно просто вспомнить прошлый семестр и увидеть обе реакции при написании контрольных и заданий.

Трудоемкость обоих алгоритмов значительно меньше, чем в исходном "наивном алгоритме", и даже растет степенным образом от *длины двоичной записи* показателя, т. е. оба алгоритмы являются эффективными. Это, конечно, хорошо.

Но можно ли дополнительно их ускорить? Вопрос этот даже в нашей игушечной постановке вовсе не праздный. Представьте себе,

---

[5] Контрольный вопрос. Почему можно проводить вычисления по тем же формулам, что мы использовали для чисел?

[6] Имейте в виду, что этот порядок — небольшой, а алгоритм Штрассена — практическая процедура для "плотно заполненных матриц". В настоящее время известно много других способов быстрого умножения матриц и рекордный имеет асимптотическую оценку числа операций чуть меньше, чем $O(n^{2.373\ldots})$, но практичным и точным алгоритмом является лишь метод Штрассена. Некоторые мои коллеги считают, что **существует практическая процедура перемножения матриц с асимптотической трудоемкостью $O(n^{2+\varepsilon})$.** Такой алгоритм может (гипотетически) перевернуть весь мир вычислительной линейной алгебры. Замечу, что примерно пять лет назад в задаче поиска наилучшего алгоритма умножения матриц произошло первое за последние 25 лет продвижение. Кстати, как это ни покажется удивительным и даже неуместным в подобном контесте, связанная с этим результатом история драматична. Об этом можно почитать, например, http://www.scottaaronson.com/blog/?p=839#comments

[7] Кстати, именно для нашей задачи такое предположение не совсем оправдано, поскольку, если считать, что мы выполняем умножение "в столбик", то сложность такой операции *пропорциональна произведению длины битовых записей сомножитей*, а длина записи сомножителей растет.

что вам нужно произвести не одно, а один гугл возведений в ту же степень n (при разных основаниях a), тогда экономия даже одного умноженияя может оказаться существенной.

Отметим, что индийский алгоритм неоптимальный. Например, можно вычислита^{54} можно выислить, используя семь умножений, а "индийский алгоритм" требует $\lambda(54)+\nu(54)-1=5+4-1=8$ умножений.

Вопросы о наилучших алгоритмах вычисления степеней рассматриваются при изучении т.н. аддитивных цепочек[8].

Что же такое *аддитивная цепочка*? Это любая, начинающаяся с 1, последовательность натуральных чисел $a_0=1,a_1,\ldots,a_m$, в которой каждое число является суммой каких-то двух предыдущих чисел (или удвоением какого-то предыдущего числа). Обозначим $l(n)$ наименьшую длину аддитивной цепочки, заканчивающейся числом n. Длиной цепочки $a_0=1,a_1,\ldots,a_m$ называем число m. Например, $1,2,3,5,7,14$ минимальная цепочка для 14, т.е. $l(14)=5$.

Поскольку показатели при умножении складываются, то, по определению, наименьшее число умножений, необходимое для возведения в n-ю степень, равно $l(n)$.

Индийский алгоритм дает оценку $l(n)\le\lambda(n)+\nu(n)-1$.

Можно показать, что справедлива нижняя оценка $l(n)\ge\lambda(n)$. (Отсюда вытекает, что $l(2^n)=n$, а оптимальным для вычисления a^{2^k} является, например, последовательное возведение в квадрат.)

Более тонкие оценки можно поискать в сети или посмотреть в книге Кнута. Зачастую они доказываются непросто, а по своей точности ненамного превосходят рассмотренные выше почти очевидные оценки. Общая задача определения $l(n)$ до сих пор не решена. Не известно даже, существует ли алгоритм полиномиальной сложности[9] для вычисления функции $l(n)$. Не решены также многие другие задачи об аддитивных цепочках. Например, неизвестно, верно ли равенство $l(2^n-1)=n+l(n)-1$ (или в другом варианте $l(2^n-1)\le n+l(n)-1$). Это достаточно широко известная гипотеза Брауэра- Шольца. Другая знаменитая нижняя граница $l(n)\ge\lambda(n)+\log_2(\nu(n))$ "почти доказана": справедливо неравенство $l(n)\ge\log_2(n)+\log_2(\nu(n))-2.13$ (но с 1974 года этот результат не улучшен). Некоторые естественные гипотезы об аддитивных цепочках оказались неверными. Обо всем этом можно почитать в замечательной книге Кнута.

Наилучшая из общих верхних оценок была доказана в тридцатые годы Альфредом Брауэром и имеет вид

$$\lambda(n)\left(1+\frac{1}{\lambda(\lambda(n))}+\frac{O(\lambda(\lambda(\lambda(n))))}{(\lambda(\lambda(n)))^2}\right).$$

Она вытекает из следующей теоремы, если в ней положить $k=\lambda(\lambda(n))-2\lambda(\lambda(\lambda(n)))$.

Теорема 1 (А. Брауэр, 1939) *При $k>1$ справедливо неравенство*

$$l(n)<(1+1/k)\log_2 n+2^k.$$

Наметим доказательство этого неравенства. Представим показатель n как число в $m=2^k$-ичной системе счисления (это т.н. 2^k-арный метод): $n=d_0m^t+d_1m^{t-1}+\ldots+d_t$. Тогда можно записать следующую аддитивную цепочку (возможно, некоторые члены в ней повторяются, но это не важно, поскольку мы хотим оценить ее длину сверху). Сначала вычислим цепочку $1,2,3,\ldots,m-2,m-1$ [в исходных терминах возведения в степень это соотвествует вычислению $a,a^2,\ldots,a^{m-1}$]. (Заметим, что на самом деле важны только показатели d_j, входящие в представление n.) Затем вычислим цепочку $2d_0,4d_0,\ldots,md_0,md_0+d_1$ [это соотвествует возведению a^{d_0} в m-ю степень и умножению на a^{d_1}. Далее рассмотрим цепочку $2(md_0+d_1),4(md_0+d_1),\ldots,m(md_0+d_1),m^2d_0+md_1+d_2$ и т.д., пока не получим n. Для наглядности выпишем всю цепочку.

$$1,2,3,\ldots,m-2,m-1$$
$$2d_0,4d_0,\ldots,md_0,md_0+d_1,$$
$$2(md_0+d_1),4(md_0+d_1),\ldots,m(md_0+d_1),m^2d_0+md_1+d_2,$$
$$\ldots\ldots\ldots,$$
$$\ldots\ldots\ldots,m^td_0+m^{t-1}d_1+\ldots+d_t=n.$$

На практике, конечно, нужно вычеркнуть из нее повторяющиеся числа, а в первой строке оставить только показатели d_j (можно провести и дополнительную оптимизацию). Далее, можно показать, что длина построенной цепочки не превышает $m-2+(k+1)t$, откуда следует заключение теоремы, поскольку, по определению, $\log_2 n\ge kt$ и $2^k=m$.

Задание на третью неделю: 20.02–26.02 [0.07]
Полиномиальные алгоритмы. Класс $\mathcal{NP}$
Раздел 4 программы
Литература: [Кормен 1, Глава 36]
[Кормен 3, Глав 34], [ДПВ. Глава 8]

Задача 10. (0.01) Пусть $q(t)=t^{2017}\in\mathbb{Z}[t]$ и $a,m\in$ Z Язык $L(a,m)\subseteq$ Z определяется правилами $x_0=a\pmod m$, $x_{i+1}=q(x_i)\pmod m$. Постройте полиномиальный алгоритм, проверяющий истинность предиката: $x\stackrel{?}{\in}L(a,m)$.

Полиномиальность метода Гаусса.

Задача 11. $(0.01+0.02)$ Рассмотрим систему линейных уравнений $Ax=b$ с целыми коэффициентами, имеющую m уравнений и n неизвестных, причем максимальный модуль целых коэффициентов A,b равен h.
(i) Оцените сверху числители и знаменатели чисел, которые могут возникнуть при непосредственном применении алгоритма Гаусса

Из решения этой задачи следует, что при прямом использовании алгоритма исключения Гаусса промежуточные результаты могут в принципе расти дважды экспоненциально, и потому, в частности, метод исключений не является полиномиальным по входу в битовой арифметике. Но оказывается, что метод Гаусса можно модифицировать так, что получится полиномиальный алгоритм. Модификация заключается в эмуляции *рациональной арифметики*. Для этого каждый (рациональный) коэффициент $\frac{p}{q}$ представляется парой (p',q') взаимно простых чисел $\frac{p}{q}=\frac{p'}{q'}$. Все арифметические действия над коэффициентами моделируются действиями над соответствующими парами, а в конце каждой операции, используя алгоритм Евклида, мы принудительно добиваемся взаимной простоты числителя и знаменателя. Скажем, эмуляция сложения коэффициентов, заданных парами $(7,10)$ и $(5,6)$, состоит в вычислении пары $(7\cdot6+5\cdot10=92,\ 6\cdot10=50)$, определении НОД$(92,60)=2$ и записи ответа $(23,15)$.

Полиномиальность указанной модификации вытекает из следующего утверждения: **все элементы матриц, возникающих в методе Гаусса, являются отношением каких-то миноров исходной расширенной матрицы системы.**

Докажем это. Без ограничения общности будем считать, что ведущие элементы расположены на главной диагонали, и обозначим $(a_{ij}^{(k)})$ матрицу, полученную после k-го исключения. Также обозначим $d_1,\ldots,d_n$ элементы главной диагонали результирующей верхнетреугольной матрицы, так что $d_i=a_{ii}^{(n)}$. Пусть $D^{(k)}$— подматрица, образованная первыми k столбцами и первыми k строками *исходной матрицы системы*, а $D_{ij}^{(k)}$, $k+1\le i,j\le n$— подматрица, образованная первыми k столбцами и столбцом i и первыми k строками и строкой j, матрицы, полученной после k-го исключения. Пусть $d_{ij}^{(k)}=det(D_{ij}^{(k)})$. По определению, $det(D^{(k)})=d_{kk}^{(k-1)}$.

Ключом является следующая формула: $a_{ij}^{(k)}=\frac{d_{ij}^{(k)}}{det(D^{(k)})}$, поскольку, в соответствии с процедурой исключений $d_{ij}^{(k)}=$

[8]Идущий ниже текст основан на компиляции текста Кнута и популярного текста.

[9]Это означает, как мы уже понимаем, что время работы алгоритма ограничено некоторым полиномом от $\log n$.

$d_1 \dots d_k a_{ij}^{(k)}$ и $det(D^{(k)}) = d_1 \dots d_k$. Таким образом, можно все время работать с дробями, числители и знаменатели которых *являются минорами* **исходной** *матрицы*, так что длина записи остается полиномиальной[10], а все вычисления по методу Гаусса (включая, конечно, вычисления НОД получаемых дробей) будут также полиномиальными.

(*ii*) Оцените трудоемкость модифицированного метода Гаусса в виде формулы от $m, n \log h$. Трудоемкость алгоритма Евклида считайте линейной по длине входа. Покажите, что модифицированный алгоритм будет полиномиальным по входу.

В следующей задаче мы установим, что многие стандартные алгоритмы курса ТРЯП являются полиномиальными.

Ниже используются следующие обозначения: NA — недетерминированный КА (НКА); DA — детерминированный КА (ДКА); N — магазинный автомат, принимающий по пустому стеку; F — магазинный автомат, принимающий по финальному состоянию; G — контекстно- свободная грамматика (КСГ).

$L(\cdot)$, где вместо "$\cdot$" можно подставить один из перечисленных выше объектов, обозначает класс языков, принимаемых (порождаемых) объектами указанного класса. Например, $L(G)$ означает класс всех КС-языков, причем описание каждого языка дается соответствующей КСГ.

Будем считать, что объекты задаются своими стандартными описаниями, например, НКА кодируется его диаграммой.

Если специально не оговорено, то предполагается, что все языки заданы в двоичном алфавите $\{0, 1\}$.

В таблице ниже строки отвечают предикатам, а столбцы — классам языков, которые задаются указанными объектами. Рассматриваются следующие предикаты[11]:

$L(\cdot) \stackrel{?}{=} \emptyset$ — язык пуст; $L(\cdot) \stackrel{?}{=} \infty$ — язык бесконечен; $w \stackrel{?}{\in} L(\cdot)$ ($w \stackrel{?}{\notin} L(\cdot)$) слово "$w$" принадлежит (не принадлежит) языку $L(\cdot)$.

Нашей целью является оценка сложности получаемых «задач», т. е. оценка сложности вычисления указанных предикатов на соответствующих классах языков. Например, ячейке (2,2) в таблице отвечает задача проверки непустоты языка, заданного ДКА.

Задача Д–6. ($2 \times 0.01 + 3 \times 0.02$) Покажите, что в колонках 2–6 таблицы можно поставить знак $\mathcal{P}$.

	DA	NA	G	N	F
$L(\cdot) \stackrel{?}{=} \emptyset$					
$L(\cdot) \stackrel{?}{=} \infty$					
$w \stackrel{?}{\in} L(\cdot)$					
$w \stackrel{?}{\notin} L(\cdot)$					

Для обоснования можно сослаться на конкретный алгоритм из курса ТРЯП и (это обязательно!) привести аргументы, показывающие, что этот алгоритм полиномиальный. Для этой задачи может оказаться полезной книга **[ХМУ]**.

Из определения почти очевидно, что класс $\mathcal{P}$ как множество языков замкнут относительно стандартных операций над языками: объединения, пересечения, дополнения, конкатенации. Несколько труднее установить, что $\mathcal{P}$ замкнут также относительно итерации.

Задача 12. (0.02) Покажите, что класс $\mathcal{P}$ замкнут относительно $*$-операции Клини ($L^* = \varepsilon \cup L \cup L^2 \cup \dots$).

Немного теории: класс $\mathcal{NP}$

Класс $\mathcal{P}$ составляют предикаты (= свойства, языки), которые можно проверить с помощью полиномиального по входу алгоритма и именно свойства свойства таких языков изучаютя в этом задании. Но Класс $\mathcal{NP}$ составляют предикаты, имеющие полиномиальные по входу сертификаты (доказательства). Формально, предикат L принадлежит классу $\mathcal{NP}$, если он представим в форме $L(x) = \exists\, y\, [(|y| < poly(|x|)) \wedge R(x, y)]$, где $R(\cdot, \cdot) \in \mathcal{P}$.

Например, пусть предикат $R(x, y) =$ "y есть гамильтонов[12] цикл в графе x". Более точно можно сказать так: "x есть двоичный код некоторого графа, а y — код гамильтонова цикла в этом графе (используем такое кодирование, при котором код цикла не длиннее кода графа), такие что…". Возьмём $poly(n) = n$. Тогда $L(x)$ в точности означает, что в графе x есть гамильтонов цикл.

Слово y понимается как "подсказка", "сертификат (= доказательство)" наличия свойства.

Рассмотрим игровую интерпретацию приведенного определения $\mathcal{NP}$.

Имеются два персонажа: король **А**ртур, умственные способности которого полиномиально ограничены, и волшебник **М**ерлин, который интеллектуально всемогущ и знает правильные ответы на все вопросы. Король **А** интересуется некоторым свойством $L(x)$ (например, "есть ли у графа x гамильтонов цикл"). Волшебник же **М** *пристрастен* и хочет, чтобы король признал наличие этого свойства (ну, скажем, граф стремится к званию гамильтонова и дал **М** взятку). **А** не доверяет своему волшебнику, зная его корыстолюбие (на родном языке короля это, видимо, звучало бы так:"He is too clever to be honest"[13]), и хочет иметь возможность самостоятельно проверить предложенный **М** ответ.

Поэтому они действуют следующим образом. **А** и **М** оба изучают "личное дело" графа (его кодировку) x, после чего **М** сообщает некоторую информацию (слово y), которая должна убедить **А**, что $L(x) = 1$. Используя эту информацию, **А** проверяет убедительность аргументов **М** некоторым полиномиальным алгоритмом $R(x, y)$.

В этих терминах определение класса $\mathcal{NP}$ можно сформулировать так: свойство L принадлежит классу $\mathcal{NP}$, если у **А**ртура есть такой полиномиальный способ $R(\cdot, \cdot)$ проверки убедительности доводов **М**ерлина, что при $L(x) = 1$ у **М** есть сертификат y, $|y| < poly(|x|)$, убеждающий **А** (т.е. $R(x, y) = 1$). А если $L(x) = 0$, то как бы **М** ни изощрялся, **А** не поверит, что $L(x) = 1$, т.к. при любом сертификате $R(x, y) = 0$.

Эквивалентно $\mathcal{NP}$ можно определить, как класс языков, распознаваемых *недетерминированными* полиномиальными МТ. Такая машина совершает полиномиальное по длине входа число переходов, но на каждом шаге может сама выбирать синтаксически допустимый переход. Как уже отмечалось в доисторические времена, механизм индетерминизма в природе, вроде бы, не наблюдается и вводится *специально* для того, чтобы хоть что-то узнать о предположительной сложности задач (= языков).

Полиномиальная *сводимость* по Карпу (или *полиномиальная many-to-one*-сводимость или *полиномиальная mapping*-сводимость) определяется следующим образом. Предикат L_1 полиномиально сводится к предикату L_2 (обозначение $L_1 \leq_p L_2$), если существует такая функция $f(\cdot)$, вычислимая некоторым *полиномиальным алгоритмом*, что $\forall x \big(x \in L_1 \Leftrightarrow f(x) \in L_2\big)$.

Из этого определения легко вытекает, что если $L_1 \leq_p L_2$, то справедливы следующие импликации: (*i*) $L_2 \in \mathcal{P} \Rightarrow L_1 \in \mathcal{P}$; (ii) $L_1 \notin \mathcal{P} \Rightarrow L_2 \notin \mathcal{P}$; (iii) $L_2 \in \mathcal{NP} \Rightarrow L_1 \in \mathcal{NP}$.

Предикат $L \in \mathcal{NP}$ называется *$\mathcal{NP}$-полным*, если любой предикат из $\mathcal{NP}$ к нему полиномиально сводится.

Самым известным $\mathcal{NP}$-полным предикатом является (согласно теореме Кука-Левина) предикат Выполнимость: $SAT(x) = 1 \Longleftrightarrow$

[10] Контрольные вопросы: почему? Можете ли вы привести оценки?

[11] Обратите, пожалуйста, внимание на то, что предикаты в некоторых строках дополнительны, т. е. являются отрицаниями друг друга. Это отражает одну из особенностей мира алгоритмов, в котором ответы «Да» и «Нет» принципиально несимметричны. Каждый понимает, что ситуация, когда программа остановилась или когда она еще работает, существенно отличаются, и поэтому, например, есть предикаты, для которых быстрые алгоритмы известны для проверки выполнимости, а для проверки невыполнимости — нет (и наоборот). Знаете ли вы такие предикаты?

[12] Простой (несамопересекающийся) замкнутый обход вершин графа. Назван так по имени того самого У.Р. Гамильтона (теорема Гамильтона-Кэли, уравнения Гамильтона и т.д.).

[13] Предложил А.Шень.

x есть формула[14] с булевыми переменными и символами ($\neg$, $\vee$, $\wedge$), которая *иногда* истинна (обычно в качестве формулы рассматривается просто конъюнктивная нормальная форма, причем можно считать, что *в каждый дизъюнкт входит не более 3 переменных или их отрицаний*, т.н. 3-КНФ предикат).

$\mathcal{NP}$-полные предикаты — самые сложные в классе $\mathcal{NP}$: если некоторый $\mathcal{NP}$-полный предикат можно вычислять за время $T(n)$, то любой предикат $L \in \mathcal{NP}$ можно вычислять за время $T(n^c)$ для некоторого фиксированного числа $c(L)$.

Задача 13. (0.01) Покажите, что класс $\mathcal{NP}$ замкнут относительно $*$-операции Клини. Укажите, как построить для результирующего языка L^*, $L \in \mathcal{NP}$ соответствующий сертификат "y" и проверочный предикат $R(x, y)$.

Задача 14. (0.01) Покажите, что классу $\mathcal{NP}$ принадлежит язык несовместных систем линейных уравнений с целыми коэффициентами от 2017 неизвестных, и постройте соответствующий сертификат "y" и проверочный предикат $R(x, y)$.

Задание на четвертую неделю: 27.02–5.03. [0.1]
Классы $\mathcal{NP}$ и $co-\mathcal{NP}$
Полиномиальная сводимость
Раздел 4 программы
Литература: [Кормен 1, Глава 36] [Кормен 3, Глава 34], [ДПВ, Глава 8]

Задача 15. (0.02) Покажите, что классу $\mathcal{NP}$ принадлежит язык несовместных систем линейных неравенств с целыми коэффициентами от 2017 неизвестных и постройте соответствующие сертификаты "y" и проверочные предикаты $R(x, y)$.

Задача Д–7. (0.03) Язык из предыдущей задачи при дополнительном предположении, что переменные $\{x_i\}$ целочисленные. Если эта задача представляет трудности, то разберите случай, когда неизвестных два ($k = 2$).

Приведем теперь пример языка, принадлежность которого классу $\mathcal{NP}$ совершенно не очевидна.

Сначала вспомним кое-какие элементарные сведения о поле вычетов $\pmod p$ Просто понять, что в $\mathcal{NP}$ лежит язык составных чисел $A = \{1, 4, 6, 8, 9, 10, \ldots\}$ (сертификатом служат предъявляемые сомножители). Но оказывается, что в $\mathcal{NP}$ лежит и язык $B = \mathrm{Z} \setminus A = \{2, 3, 5, 7, 11, \ldots\}$ простых чисел[15]. Полиномиальный сертификат устроен хитро. Как мы знаем, $p \in B \Leftrightarrow \exists g : \{g^i \pmod p,\ i = 1, 2, \ldots, p-1\} = \{1, 2, \ldots, p-1\}$ (написано равенство множеств). Поскольку длина записи числа p составляет $\log p$, то длина сертификата должна быть $\mathrm{poly}(\log p)$. И если быстро возводить числа $\pmod p$ в степень мы еще умеем[16], то все равно массив $\{g^i \pmod p\}$ слишком длинный. Но, как мы помним, вычет g с нужными свойствами существует тогда и только тогда, когда выполнено $g^{\frac{p-1}{p_1}} \neq 1 \pmod p, \ldots, g^{\frac{p-1}{p_k}} \neq 1 \pmod p$, где $p_1, \ldots, p_k$ — это все простые делители числа $p - 1 = p_1^{l_1} \ldots p_k^{l_k}$. Число проверок действительно уменьшилось и стало полиномиальным (их заведомо не больше $\log p$), но, кажется, что мы ничего не выиграли: нам ведь нужно решить **ту же задачу построения сертификата простоты** для всех p_j, $j = 1, 2, \ldots, k$. Хитрость заключается в том, что нужно применить ту же идею рекурсивно, поскольку длина сертификатов для всех p_i сильно уменьшилась! Фактически сертификатом будет дерево с нужными пометками в вершинах, и нам нужно показать, что суммарная длина всех участвующих в описании дерева компонентов останется полиномиальной по $\log p$.

Задача 16. (0.02) Постройте NP-сертификат простоты для числа $p = 3911$, $g = 13$. Простыми в рекурсивном построении считаются только числа $2, 3, 5$ (они сами являются своими сертификатами).

Задача 17. (0.01) Покажите, что язык *разложения на множители (факторизации)* $L_{factor} = \{(N, M) \in \mathrm{Z}^2 \mid 1 < M < N$ и N имеет делитель d, $1 < d \leq M\}$ принадлежит $\mathcal{NP} \cap co-\mathcal{NP}$.

Полиномиальная сводимость.

Полиномиальная *сводимость* по Карпу (или *полиномиальная many-to-one*-сводимость или *полиномиальная mapping*-сводимость). Предикат L_1 полиномиально сводится к предикату L_2 (обозначение $L_1 \leq_p L_2$), если существует такая функция $f(\cdot)$, вычислимая некоторым *полиномиальным алгоритмом*, что $\forall x \big(x \in L_1 \Leftrightarrow f(x) \in L_2\big)$.

Из этого определения легко вытекает, что если $L_1 \leq_p L_2$, то справедливы следующие импликации: (i) $L_2 \in \mathcal{P} \Rightarrow L_1 \in \mathcal{P}$; (ii) $L_1 \notin \mathcal{P} \Rightarrow L_2 \notin \mathcal{P}$; (iii) $L_2 \in \mathcal{NP} \Rightarrow L_1 \in \mathcal{NP}$.

Хочется отметить, что определение сводимости нетривиальное, и с его непониманием связано большинство ошибок. Например, попробуйте сразу, исходя из определения, ответить на совершенно пустяковый вопрос: правда ли, что полиномиальная сводимость взаимно- однозначная?

Неформально сводимости упорядочивают различные языки по сложности. Действительно, утверждение о том, что язык $\mathcal{A}$ полиномиально сводится (по Карпу) к языку $\mathcal{B}$, по определению, означает, что выяснить принадлежность произвольного слова $w \stackrel{?}{\in} \mathcal{A}$ можно за полиномиальное по длине $|w|$ время, если в конце вычисления можно обратиться к подпрограмме- оракулу, которая может определить принадлежность языку $\mathcal{B}$ произвольных не слишком длинных слов (длины не более $\mathrm{poly}(|w|)$). Иначе говоря, процедура распознавания слова $w \stackrel{?}{\in} \mathcal{A}$ заключается в том, что мы сначала за полиномиальное (по длине $|w|$) время преобразуем w в некоторое слово u, такое что $|u| \leq \mathrm{poly}(|w|)$, $w \in \mathcal{A} \Leftrightarrow u \in \mathcal{B}$, что неформально означает, что распознавание языка $\mathcal{B}$ во всяком случае не легче распознавания языка $\mathcal{A}$. Следующая задача является иллюстрацией этих соображений.

Заданы n точек плоскости V с координатами $\{(x_1, y_1), \ldots (x_n, y_n)\}$. Требуется найти их *выпуклую оболочку*, т. е. наименьшее по включению выпуклое множество S_V, такое что $V \subseteq S_V$. Рассмотрим следующую модель вычислений, в которой за единицу времени можно выполнять следующие операции: 1) сравнение двух чисел; 2) сложение чисел; 3) возведение числа в квадрат (т. е. вычисление x^2 по заданному x).

Задача Д–8. (0.02) В описанной модели вычислений задача сортировки n чисел за **линейное** время сводится к задаче построения выпуклой оболочки n точек плоскости.

Таким образом, задача сортировки оказывается **не сложнее** задачи построения выпуклой оболочки. Более того, если доказать, что в рассматриваемой модели нижняя оценка сортировки $\Omega(n \log n)$ сохраняется[17], то мы получим оптимальную $\Omega(n \log n)$ **нижнюю оценку** для построения выпуклой оболочки, поскольку алгоритмы, имеющие такую трудоемкость, известны.

[14] Кстати, здесь самое время вспомнить **формальное** определение формулы.

[15] В 2002 году появилась сенсационная работа, показывающая, что $B \in \mathcal{P}$. Если вы будете ссылаться на ее результаты, то **обязаны** привести доказательство.

[16] Контрольные вопросы. Каким образом? Обуждалась ли такая задача ранее?

[17] Мы вернемся к этому утверждению, когда будем рассматривать сложность сортировки попарными сравнениями.

Задача 18. (0.03) Язык ГП состоит из всех графов, имеющих гамильтонов путь (несамопересекающийся путь, проходящий через все вершины графа). Язык ГЦ состоит из всех графов, имеющих гамильтонов цикл (цикл, проходящий через все вершины, в котором все вершины, кроме первой и последней, попарно различны). Постройте **явные** полиномиальные сводимости ГП к ГЦ и наоборот. Графы задаются матрицей смежности.

В курсе ТРЯП мы построили полиномиальный алгоритм проверки неэквивалентности регулярных языков, заданных ДКА. А какова сложность вычисления этого предиката, если один или оба языка, представлены НКА?

Задача Д–9. (0.03) Покажите, что задача проверки неэквивалентности регулярных языков, заданных НКА, в унарном (однобуквенном) алфавите принадлежит классу $\mathcal{NP}$.

Граф $G(V,E)$ называется *планарным*, если его можно вложить в плоскость, т. е. существует отображение $f: G \to \mathrm{R}^2$, посылающее вершины V в различные точки плоскости, а ребра E — в кривые (которые можно считать ломаными), соединяющие соответствующие точки-вершины, при этом кривые-ребра имеют право пересекаться только по вершинам. Соответственно, граф $G(V,E)$ называется *торическим*, если его можно вложить в *тор* — поверхность бублика — $T = S^1 \times S^1$ (S^1 — это стандартное обозначение одномерной сферы — окружности). По определению, любой планарный граф является торическим, но не наоборот. Как известно, на плоскости нельзя нарисовать без пересечения ребер (вложить) ни граф K_5 (полный пятивершинник), ни граф $K_{3,3}$ (три дома, три колодца), но их можно вложить в тор, поскольку на плоскости удается изобразить все ребра этих графов, кроме одного, которое можно пустить по ручке (вдоль параллели тора). Априори не очень понятно, удастся ли такой трюк с графом K_6 (полный 6-вершинник) или, тем более, с графом K_7 (полный 7-вершинник).

Задача Д–10. (0.01) Докажите или опровергните, что граф K_7 является торическим. Тор удобно изображать прямоугольником с отождествленными противоположными сторонами.

Задача Д–11. (0.02) Покажите, что класс торических графов принадлежит $\mathcal{NP}$.

Задание на пятую неделю: 6.03–12.03. [0.1]
Классы $\mathcal{NP}$ и $co-\mathcal{NP}$
NP-полные языки. Полиномиальная сводимость
Раздел 4 программы
Литература: [Кормен 1, Глава 36] [Кормен 3, Глава 34], [ДПВ, Глава 8]

Самый известный NP-полный язык — это, конечно, ВЫПОЛНИМОСТЬ, который состоит из кодировок всех выполнимых булевых формул. Иначе говоря, для каждой формулы [18] из языка ВЫПОЛНИМОСТЬ существуют такие значения переменных, при которых эта формула истинна. Можно считать формулы не произвольными, а, например, КНФ или даже 3-КНФ, у которых в каждый дизъюнкт входит не более 3 переменных. В последнем случае получаем язык 3-ВЫПОЛНИМОСТЬ. Можно дополнительно предполагать, как это делается в **[Кормен 1]** или **[Кормен 2]**, что *в каждый дизъюнкт входит ровно три литерала* и что *все литералы в каждом дизъюнкте 3-КНФ различны*. Но от этого требования можно и отказаться, если окажется проще строить какие-то сводимости, т. е. рассмотреть более широкий полный язык, в котором литералы в дизъюнктах могут повторяться и в каждый дизъюнкт входит не более трех литералов. **Такой трактовки языка** 3-ВЫПОЛНИМОСТЬ **мы и будем придерживаться в этом задании**. Тогда при часто используемом преобразовании 3-КНФ в РОВНО-3-КНФ можно просто дополнить дизъюнкт нужным числом литералов. Например, дизъюнкт $\neg x_2 \vee x_3$ переписывается в эквивалентном виде $\neg x_2 \vee x_3 \vee x_3$ или $\neg x_2 \vee x_3 \vee \neg x_2$. Другое дело, что некоторые сводимости при таком понимании 3-КНФ, возможно, перестанут выполняться, и тогда нужно уточнить и/или изменить сами сводимости.

Приведем несколько примеров NP-полных языков.

ПРОТЫКАЮЩЕЕ МНОЖЕСТВО Дано семейство конечных множеств $\{A_1, \ldots, A_m\}$ и натуральное число k. Существует множество мощности k, пересекающее *каждое* A_i. Язык остается NP-полным, *даже если предположиить, что мощности всех* A_i *равны* 2.

КЛИКА. Даны неориентированый граф G и натуральное число k. В G есть клика (полный подграф) на k вершинах.

ВЕРШИННОЕ ПОКРЫТИЕ

ХРОМАТИЧЕСКОЕ ЧИСЛО. Даны неориентированый граф G и натуральное число k. Вершины G можно раскрасить в k цветов так, чтобы смежные вершины были окрашены в разные цвета. При $k = 3$ получаем язык 3-COLOR и он также NP-полный.

ГАМИЛЬТОНОВ ГРАФ. Дан неориентированый граф G, в котором есть *гамильтонов цикл*. Иными словами, существует циклический обход всех вершин графа, не попадающий ни в какую вершину дважды.

РАЗБИЕНИЕ или ЗАДАЧА О КАМНЯХ Дано конечное множество (куча) камней A, причем вес каждого камня $a \in A$ является целым положительным числом $s(a)$. Можно разбить A на две кучи одинакового веса. Иными словами, существует такое подмножество $A' \in A$, что $\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$.

3-СОЧЕТАНИЕ. Дано множество $M \subseteq W \times X \times Y$, где W, X и Y — непересекающиеся множества, содержащие одинаковое число элементов q. В M есть *трехмерное сочетание*, т. е. такое подмножество $M' \subseteq M$ мощности q, никакие два элемента которого не имеют ни одной одинаковой координаты.

РЮКЗАК. Дана натуральные числа $\{a_1, \ldots, a_n\}$ и натуральное число b, такие что сумма некоторых a_i равна b.

max −2-ВЫПОЛНИМОСТЬ. Дана 2-КНФ (т. е. КНФ, в каждую дизъюнкцию которой входит не более двух логических переменных) и двоичное число k. Существует такой набор значений логических переменных, что выполняются k или более дизъюнкций.

МАКСИМАЛЬНЫЙ РАЗРЕЗ. Дан граф G и натуральное число k. Множество вершин графа можно разбить на два непересекающихся подмножества, между которыми можно провести не менее k ребер.

Иногда говорят о взвешенном варианте задачи. Дан граф $G(V,E)$ с неотрицательной весовой функцией на ребрах $w: E \to \mathrm{Z}_+$ и натуральное число k. Можно найти дизъюнктное разбиение множества $V = V_1 \sqcup V_2$, такое что сумма весов ребер, соединяющих V_1 и V_2, не менее k.

N[от]A[ll]E[qual]-SAT. Дана КНФ-формула, для которой существует набор, такой что в каждом дизъюнкте есть истинный и ложный литералы.

Зафиксируем выполнимую КНФ $\psi(x_1, x_2, x_3) = (x_1 \vee x_2 \vee \neg x_3)$ [зависящую от трех переменных и имеющую 1 дизъюнкт] и НЕвыполнимую КНФ $\chi(x_1, x_2) = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge \neg x_1$ [зависящую от двух переменных и имеющую 3 дизъюнкта].

Везде ниже мы будем иллюстрировать сводимости, используя именно эти КНФ.

Задача 19. (0.01) В **[Кормен 1]** или **[Кормен 2]** предполагается, что в языке 3-ВЫПОЛНИМОСТЬ (по Кормену) в каждый дизъюнкт входит ровно три литерала и все литералы в каждом дизъюнкте различны. Укажите, как за полиномиальное время преобразовать

[18] Проверьте себя: приведите формальное определение **булевой формулы**. Чем отличается **булева схема** от **булевой формулы**?

произвольную 3-КНФ ϕ, в которой в каждом дизъюнкте содержится не более трех литералов, причем литералы могут повторяться, в РОВНО-3-КНФ $\tilde{\phi}$, в которой в каждый дизъюнкт входит РОВНО три неповторяющихся литерала. При этом ϕ должна быть выполнима тогда и только тогда, когда выполнима $\tilde{\phi}$. Иными словами, постройте полиномиальную сводимость языка 3-ВЫПОЛНИМОСТЬ к языку 3-ВЫПОЛНИМОСТЬ (по Кормену).

Задача 20. (2×0.01) Постройте сводимость языка ВЫПОЛНИМОСТЬ к языку ПРОТЫКАЮЩЕЕ МНОЖЕСТВО.

Конструкция такова. Пусть $\phi(x_1,\ldots,x_n)$ КНФ. Построим по КНФ семейство подмножеств $\mathcal{A}_\phi$ базового множества $\{x_1,\ldots,x_n,\neg x_1,\ldots,\neg x_n\}$. Во-первых, включим в $\mathcal{A}_\phi$ n подмножеств вида $A_i=\{x_i,\neg x_i\}$, $i=1,\ldots,n$. Во-вторых, для каждого дизъюнкта C, входящего в $\phi(\cdot)$, добавим к $\mathcal{A}_\phi$ подмножество A_C, состоящее из всех входящих в C логических переменных (если в C входит логическая переменная x_i, то включаем в A_C элемент x_i, а если в C входит переменная $\neg x_i$,то включаем в A_C элемент $\neg x_i$).

Для обоснования сводимости нужно доказать, что **исходная КНФ $\phi(\cdot)$ выполнима тогда и только тогда, когда $\mathcal{A}_\phi$ имеет протыкающее множество мощности n.** Обоснование легко получить, если решить две следующие задачи.

(i) Укажите для семейства $\mathcal{A}_\psi$ соответствующее **трехэлементное** протыкающее множество.
(ii) Докажите, что мощность любого протыкающего множества для семейства $\mathcal{A}_\chi$ больше двух.

Если использовать полный язык 3-ВЫПОЛНИМОСТЬ, то из построенной сводимости следует, что язык остается NP-полным, даже если все A_i имеют не более 3 элементов. Но оказывается, что язык остается NP-полным, даже если все A_i двухэлементные. Если отождествить эти пары элементов с ребрами некоторого графа, то соответствующий язык известен как ВЕРШИННОЕ ПОКРЫТИЕ: даны неориентированый граф $G=(V,E)$ и натуральное число k. о В G есть *вершинное покрытие* мощности k, т. е. такое подмножество вершин $V'\subseteq V$ мощности k, что хотя бы *один конец каждого ребра* входит в V'. Покажем, что этот язык также NP-полон. Для этого сведем к нему язык 3-ВЫПОЛНИМОСТЬ[19]. Во-первых, будем считать, что исходная КНФ дополнена до РОВНО-3-КНФ и в каждый ее дизъюнкт входит ровно три литерала. Построим по КНФ $\phi(x_1,\ldots,x_n)$ граф G_ϕ, вершины которого помечены и делятся на *литеральные* и *дизъюнктные*. Для каждой логической переменной x_i образуем пару **смежных** литеральных вершин, помеченных, соответственно, x_i и $\neg x_i$. Для каждого 3-дизъюнкта C образуем три **смежных** дизъюнктных вершины, помеченных переменными этого дизъюнкта. Каждую дизъюнктную вершину соединим с соответствующей литеральной вершиной, имеющей ту же метку. Если ϕ имела m дизъюктов, то, по построению, G_ϕ имеет $2n+3m$ вершин.

Для обоснования сводимости нужно доказать, что ϕ **выполнима, если и только если G имеет вершинное покрытие мощности $\mathbf{n}+2m$.** Обоснование может быть построено, если решить следующую задачу.

Задача 21. (2×0.01) (i) Укажите для графа G_ψ соответствующее $(n_{new}(\psi)+2m_{new}(\psi))$-вершинное покрытие.
(ii) Докажите, что мощность любого вершинного покрытия для графа G_χ больше $(n_{new}(\chi)+2m_{new}(\chi))$.

Здесь $n_{new}(\cdot)$, $m_{new}(\cdot)$ обозначают, соответственно, число переменных и число дизъюнктов КНФ после ее преобразования в РОВНО-3-КНФ.

[19] В книге **[Кормен 1, §36.5.2]** строится другая сводимость, использующая NP-полный язык КЛИКА.

В **[Кормен 1, §36.5.1]** или **[Кормен 2, §34.5.1]** описано построение по любой РОВНО-3-КНФ $\phi(x_1,\ldots,x_n)$ с m дизъюнктами графа $\tilde{G}_\phi$ на $3m$ вершинах, в котором имеется клика размера m тогда и только тогда, когда $\phi(x_1,\ldots,x_n)$ выполнима. Следующая задача посвящена этой сводимости. Конструкция такова. Каждому дизъюнкту отвечает тройка вершин-переменных, а ребро соединяет вершины u и v тогда и только тогда, когда они приписаны разным дизъюнктам, а отвечающие им переменные не являются отрицанием друг друга. Следующая задача посвящена этой сводимости. Сначала ψ и χ нужно преобразовать в РОВНО-3-КНФ, которые содержат m и n 3-дизъюнктов, соответственно.

Задача 22. (2×0.01) (i) Укажите для графа $\tilde{G}_\psi$ соответствующую m-клику.
(ii) Докажите, что мощность любой клики в графе $\tilde{G}_\chi$ меньше n.

О NP-полноте языков ГАМИЛЬТОНОВ ГРАФ и РАЗБИЕНИЕ см.: **[Кормен 1, §36.5.4]** и **[Кормен 1, задача 36.5-4]** (соответственно, **[Кормен 2, §34.5.3]** и **[Кормен 1, задача 34.5-5]**).

Опишем полиномиальную сводимость NP-полного языка 3-ВЫПОЛНИМОСТЬ к языку $\max-2$-ВЫПОЛНИМОСТЬ (этим будет доказана полнота языка $\max-2$-ВЫПОЛНИМОСТЬ в $\mathcal{NP}$, поскольку его принадлежность $\mathcal{NP}$ очевидна).

Сначала преобразуем 3-КНФ в эквивалентную 3-КНФ, в которой каждая дизъюнкция содержит в точности 3 переменные. Для любой 3-КНФ $\alpha=\bigwedge_1^n(a_i\vee b_i\vee c_i)$, где a_i,b_i,c_i — это либо некоторая логическая переменная, либо ее отрицание, построим 2-КНФ y следующим образом: для i-й дизъюнкции $(a_i\vee b_i\vee c_i)$ включим в y 10 следующих дизъюнкций: $L_i=\{a_i,b_i,c_i,d_i,a_i\vee\neg b_i,\neg a_i\vee\neg c_i,\neg b_i\vee\neg c_i,\neg a_i\vee\neg d_i,b_i\vee\neg d_i,c_i\vee\neg d_i\}$, где d_i, $i=1,\ldots,n$ — это новые логические переменные. Таким образом, осталось проверить, что если i-я дизъюнкция выполнима [в 3-КНФ], то можно так подобрать значение переменной d_i, что не менее q дизъюнкций из L_i будут выполнимы. А если i-я дизъюнкция невыполнима [в 3-КНФ], то при любом значении переменной d_i, меньше q дизъюнкций из L_i будут выполнимы. (q — это параметр, который вы должны найти самостоятельно.) Таким образом, если исходная 3-КНФ α выполнима, то в 2-КНФ $\bigwedge_1^n L_i$ будет выполнено не менее qn 2-дизъюнктов. И наоборот, для любой невыполнимой 3-КНФ α в 2-КНФ $\bigwedge_1^n L_i$ менее qn дизъюнктов будет выполнено.

Задача 23. (2×0.02) (i) Преобразуйте ψ в РОВНО-3-КНФ [в которой образовалось k 3-дизъюнктов] и вычислите результирующую 2-КНФ $\tilde{\psi}$ при указанной полиномиальной сводимости, указав пороговое значение kq.

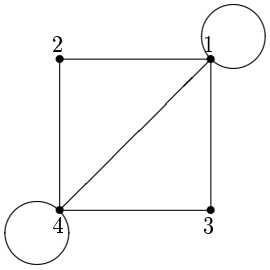
(ii) Укажите какой-нибудь набор значений логических переменных, при которых в $\tilde{\psi}$ выполнено $\geq kq$ дизюнктов.

Задача 24. (0.02) Покажите, что если язык 3-COLOR$\in\mathcal{P}$, то за полиномиальное время можно не только определить, что граф допускает раскраску вершин в три цвета, но и найти какую-то 3-раскраску (если она существует). *Обратите внимание, что на вход процедуры, проверяющей 3-раскрашиваемость, нельзя подавать частично окрашенные графы.*

Задание на шестую неделю: 13.03–19.03. [0.1]
Теоретико-числовые алгоритмы
Раздел 6 программы
Литература: [Кормен 1, Глава 33]
[Кормен 2, Глава 31], [ДПВ, Глава 2]

[Виноградов]

Задача 25. (4×0.02) Ср. **[Кормен 1, задача 33.3]**. Диаграмма графа G изображена на рисунке.



Путь в графе — это произвольная последовательность смежных вершин (возможны возвраты): $c = \{v_1, \ldots, v_l\}$. По определению, длина пути c равна $l - 1$.

Пусть g_n — это число путей в G длины n, которые начинаются в вершине 1. Из определения следует, что $g(0) = 1$ (единственный путь: $0 \to 0$), а $g(1) = 4$ (пути: $1 \to 1$, $1 \to 2$, $1 \to 4$, $1 \to 3$).

Пусть A — это матрица инциденций графа G:

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}.$$

(*i*) Вычислите число $g(2)$ путей в G длины 2 и проверьте, что оно равно сумме элементов первой строки матрицы A^2. Объясните это совпадение и докажите общую формулу для $g(n)$.

(*ii*) Найдите рекуррентное соотношение, которому удовлетворяет последовательность $\{g_n,\ n = 0, 1, \ldots\}$.

Подсказка. В ответе должна получиться рекуррентность с целыми коэффициентами типа рекуррентности Фибоначчи: $g_{n+2} = Pg_{n+1} + Qg_n$. *Можно просто подобрать коэффициенты и* **доказать**, *что они искомые. При этом необходимо вычислить хотя бы еще одно значение* $g(n)$.

Рассмотрим два способа вычисления $\{g_n,\ n = 0, 1, \ldots\}$.

Первый, основан на том, что последовательность $\{g_n\}$ удовлетворяет рекуррентному соотношению, и это подсказывает следующий *матричный* способ ее вычисления. Имеет место матричная формула[20] $\begin{pmatrix} g_n \\ g_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ Q & P \end{pmatrix}^{n-1} \begin{pmatrix} g_1 \\ g_2 \end{pmatrix}$

При вычислении $\{g_n\}$ этим способом можно использовать быстрое, например, "индийское возведение в степень" n за $O(\log n)$ тактов[21].

Второй способ вычислений основан на явном аналитическом решении линейной рекуррентности, которое можно получить самостоятельно или воспользоваться алгоритмом из текста на сайте. Например, для чисел Фибоначчи ($F_1 = 1, F_2 = 1, \ldots F_{n+2} = F_{n+1} + F_n$)) этот способ приводит к известной формуле Бине: $F_n = \frac{(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n}{\sqrt{5}}$. У вас должна получится похожая формула, также содержащая квадратичную иррациональность $\sqrt{d} = \sqrt{P^2 + 4Q}$. Вычисление по аналитической формуле реализуется чуть хитрее, чем по матричной. Для каждого значения n нужно специфицировать операции и указать с какой точностью их нужно проводить (например, для вычисления $\sqrt{d}$ нужно указать процедуру вычисления, задать точность вычисления и оценить битовую трудоемкость процедуры).

На самом деле, переход к собственным векторам матрицы $\begin{pmatrix} 0 & 1 \\ Q & P \end{pmatrix}$ *показывает, что оба алгоритмы тесно связаны, и матричный алгоритм можно рассматривать как корректный способ округления ответа, полученного по аналитической формуле.*

Оценим трудоемкость нескольких алгоритмов вычисления g_n по простому модулю p.

(*iii*) Непосредственное вычисление по рекуррентной формуле. Оцените его трудоемкость при вычислении $A = g_{20000} \pmod{29}$.

(*iv*) Докажите, что последовательность $\{g_n\}$ периодична по любому модулю. Оцените ее период для $\pmod{29}$ и найдите трудоемкость вычисления (сложность нахождения периода + сложность вычисления A) этим способом.

Задача Д–12. (3×0.02) (*i*) Пусть известно, что 5 *является квадратичным вычетом* по $\pmod p$, например, $p = 29$, т. е. разрешимо уравнение $x^2 = 5 \pmod p$. Обоснуйте алгоритм непосредственного вычисления A по аналитической формуле, т. е. прямо извлекая квадратный корень в конечном поле $\pmod p$ и проводя дальнейшие арифметические вычисления. Вычислите A этим способом. Оцените трудоемкость вычисления $g_n \pmod p$ для этого случая.

(*ii*) Пусть теперь 5 *НЕ является квадратичным вычетом* по $\pmod p$, например, $p = 23$. Придумайте и обоснуйте использующий аналитическую формулу алгоритм вычисления чисел $\{g_n\}$ по такому модулю. Проведите вычисления $A = g_{10000} \pmod{23}$. Оцените трудоемкость вычисления $g_n \pmod p$ для этого случая.

В этой задаче вам предстоит разобраться, как использовать матричный алгоритм для вычислении реккурентности по простому модулю. Мы уже знаем, что эффективность процедуры сильно (или даже критически) зависит от вычисления периода $\{g_n\} \pmod p$. И, собственно, основной вопрос, на который хочется найти ответ, как найти период в матричном представлении $\{g_n\}$. Предлагается придумать алгоритм самостоятельно и/или проанализировать следующий способ. Заметим, что нам повезло, и матрицу $\begin{pmatrix} 0 & 1 \\ Q & P \end{pmatrix}$ можно диагонализировать, т. е. привести ее к виду $S\begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}S^{-1}$, где λ_1, λ_2 — это собственные числа, а S — невырожденная матрица. Возводя в n-ю степень, получим $S\begin{pmatrix} \lambda_1^n & 0 \\ 0 & \lambda_2^n \end{pmatrix}S^{-1}$.

(*iii*) Наша задача состоит в том, чтобы обосновать эти манипуляции при вычислениях $\pmod p$ и понять, в какую степень нужно возвести матрицу, чтобы получилась единичная матрица, т. е. получить аналог малой теоремы Ферма для матриц указанного вида[22]

Оцените сложность вычисления периода и вычисления $\{g_n\} \pmod p$ матричным способом и сравните его с алгоритмами из пунктов (*i*)–(*ii*).

Комментарий. В последней задаче в пунктах (*i*)–(*ii*) речь идет о т.н. *квадратичных вычетах* и возникает естественный вопрос, как по данному числу a проверить разрешимость уравнения $x^2 = a \pmod p$ (в задаче $a = 5$, $n = 29, 23$). Можно, конечно, перебрать все вычеты, используя $O(p)$ операций, но есть и более интеллигентный полиномиальный по **длине записи** $\log p$ алгоритм, Он

[20] Поскольку для $\{g_n\}$ коэффициенты P и Q — целые, то при вычислениях можно использовать только целую арифметику.

[21] Мы разбирали этот алгоритм в первом задании, и он с необходимыми изменениями переносится на матрицы.

[22] Обратите внимания, что прямого аналога малой теоремы Ферма для матриц быть не может, поскольку, например, существуют т.н. нильпотентные матрицы (какая-то их степень равна нулевой матрице). Удивительно, но тексты, посвященные аналогам теоремы Ферма для матриц, появились только в этом тысячелетии (см., www.mathnet.ru/mp238). В принципе, их мог написать сильный студент.

основан на знаменитом *квадратичном законе взаимности*, о котором можно прочитать в книге **[Виноградов, гл. 2]** (и придумать алгоритм самостоятельно). Но есть и еще более простой способ, основанной на обобщении малой теоремы Ферма. Если определить (см. , **[Виноградов, гл. 2]**) т. н. *символ Лежандра*: $\left(\frac{a}{p}\right)$, равный $+1$, если число $a \neq 0 \pmod p$ является квадратичным вычетом $\pmod p$, и, равный -1, в противном случае, то имеет место равенство $a^{\frac{p-1}{2}} = \left(\frac{a}{p}\right) \pmod p$. Таким образом, можно эффективно проверить, является ли a квадратичным вычетом, используя быстрое возведение в степень. Кроме того, можно построить быстрый вероятностный алгоритм поиска квадратичного вычета или невычета. Что понимается по этим поясняет следующая задача.

Задача 26. (*i*) Пусть НОД$(a, N) = 1$ и $a^{N-1} \neq 1 \pmod N$. Тогда по крайней мере для половины чисел из промежутка $1 \leq b < N$ выполнено $b^{N-1} \neq 1 \pmod N$.

Комментарий. Результаты этого простого, но важнейшего упражнения позволяют строить быстрые **вероятностные** алгоритмы, проверяющие простоту чисел.

Речь идет о процедурах, которые, получив на вход n-битовую двоичную запись числа, могут быстро[23] проверять, является ли рассматриваемое число простым или составным. При этом вероятностным алгоритмам разрешается по ходу вычислений совершать переходы в зависимости от результатов бросания монетки. Тут, конечно, нужно уточнить, как следует понимать время работы вероятностного алгоритма, поскольку каждому исходу бросания монеток отвечает свое (возможно, очень длинное) вычисление. В случае алгоритмов проверки простоты речь идет о построении полиномиальных процедур типа "Лас-Вегас" (это термин), которые не ошибаются, если число составное, а если число простое, то алгоритм может выдать неправильный ответ, но вероятность этого события меньше, чем фиксированная константа, скажем $\frac{3}{4}$. Поэтому, если **независимо**[24] повторить такой Лас-Вегас алгоритм k раз и во всех случаях он выдаст ответ "простое", то с вероятностью $1-(\frac{3}{4})^k$ число действительно будет простым. При таком подходе вероятность $1-0.75^{1000}$ нужно рассматривать как практическую достоверность, и именно такими методами были на сегодняшний день построены самые большие доказуемо простые числа.

Рассмотрим один подобный алгоритм "ТЕСТ ФЕРМА".

Вход: натуральное число N.

Выход: "ДА", если N — простое;

"НЕТ", в противном случае.

Случайно выбираем положительное целое $1 < a < N$.

Если НОД$(a, N) > 1$ *То*

Выход: НЕТ

Иначе { *Если* $a^{N-1} = 1 \pmod N$ *То Выход: ДА Иначе Выход: НЕТ* }

Описанный алгоритм — это "почти" полиномиальный вероятностный алгоритм проверки простоты

(*ii*) Покажите, что "ТЕСТ ФЕРМА" может быть реализован за полиномиальное по входу число операций.

(*iii*) Пусть известно, что составное число N не является числом Кармайкла[25], т. е. для некоторого натурального a, взаимно-простого с N, выполнено $a^{N-1} \neq 1 \pmod N$, тогда "ТЕСТ ФЕРМА" выдает правильный ответ с вероятностью[26] $\geq \frac{1}{2}$.

Комментарий (продолжение). Последняя задача должна навести на мысль, что вероятность нужно рассматривать как важный **вычислительный ресурс**, иногда позволяющий существенно снизить трудоемкость. Но при этом соответственно должны усилиться требования к "источникам случайности". Уже не достаточно сослаться на какой-то текст в котором 10000 часов прогонялась функция RANDOM и построены какие-то сравнительные графики или даже диаграммы. Подумаем, почему такой стандартный в программистской практике ответ нас не удовлетворяет.

Пусть N составное. Рассмотрим множество $\mathcal{A} \subseteq \{2, 3, \ldots, N-1\}$, состоящее из всех чисел, для которых нарушается тест Ферма. Согласно предыдущей задаче $|\mathcal{A}| \geq (N-1)/2$, но мы абсолютно ничего не знаем о структуре $\mathcal{A}$. Корректный случайный генератор должен уметь обслуживать (т.е. достаточно часто попадать) **во все** такие множества $\mathcal{A}$. Поэтому термин "случайный" при формальном описании алгоритма и конкретная реализация этой случайности при написании программы несут совершенно разные смысловые нагрузки. Это может быть источником путаницы, а также формальных и фактических ошибок. Например, совершенно не понятно, почему аналоговый генератор случайных чисел, получаемый из обработки щелчков счетчика Гейгера, удовлетворяет нашим требованиям (не говоря уже о многочисленных программных реализациях функции RANDOM).

Важно понять, что когда мы апеллируем к идеальному случайному генератору мы обращаемся с сложному (в принципе, сложнейшему) вычислительному ресурсу, а в программистской практике мы заменяем его некоторой доморощенной поделкой. Почему этот трюк срабатывает? Во-первых, потому что люди, придумавшие эти поделки, прекрасно понимали суть проблемы, и поделки достаточны хороши[27], а во-вторых, —- возможно, что сами множества, которые мы исследуем с помощью простых случайных генераторов, и не такие уж сложные. Например, согласно одной недоказанной гипотезе любое множество $\mathcal{A}$ обязательно содержит и "маленькие" числа (и их можно просто найти тупым перебором)[28]. Вероятностный алгоритм, предложенный в последней задаче работает не всегда, но его можно модифицировать до корректного (об этом можно прочитать **[Кормен 1, §33.8], [Кормен 2, §31.8] или в книге [К-Ш-В]**). А в 2002 году удалось построить детерминированный полиномиальный алгоритм проверки простоты, использующий близкие идеи. В следующем недельном задании мы подробнее остановимся на нем[29].

Схема RSA. Краткий конспект

Итак, для проверки простоты удается построить вероятностные и даже детерминированные эффективные алгоритмы. А как обстоит дело с близкой задачей *факторизации*, т. е. нахождения делителей большого составного числа N? Стандартный алгоритм, т.н. "решето Эратосфена", который обсуждался на первой лекции, заключается в том, что мы мысленно составляем список всех чисел от 1 до N. Последовательно вычеркиваем из списка сначала единицу, затем двойку и все кратные ей. Затем все кратные первому оставшемуся невычеркнутому числу, т.е. тройке и т.д. пока не вычеркнем N. Если это произошло на последнем шаге, то N — простое, иначе мы находим делитель. Мы пришли к заключению, что решето не является полиномиальным алгоритмом (почему?).

С современной точки зрения задачи проверки простоты и факторизации гипотетически принципиально различны, и предположительно для последней вообще нельзя построить эффективного алгоритма[30]. Грубо говоря, ничего лучше, чем рассмотренное выше

---

[23] За полиномиальное **по длине записи** — n —- время (число операций). Помните, что само число может быть порядка 2^n.

[24] Осмысление понятия *независимости* составляет важную часть теории вероятностей.

[25] Числа Кармайкла — это составные числа, для которых тест Ферма выполняется для всех чисел a, взаимно простых с модулем N. Встречаются они редко. Везде приводится первое число Кармайкла — 561, попробуйте найти второе. Известно, что чисел Кармайкла бесконечно много, но доказан этот факт был совсем недавно. Как с ними бороться и как построить корректный полиномиальный вероятностный алгоритм проверки простоты можно прочитать в книге **[Кормен 1, §33.8]** или в книге **[К-Ш-В]**.

[26] Вероятность понимается в "наивном смысле" как отношения числа благоприятных исходов к общему числу исходов.

[27] Достаточно вспомнить, что линейный модульный генератор предложил Дж. фон Нейман.

[28] Формально говоря, речь идет о том, существует ли полилогафмическая граница на величину самого первого квадратичного невычета $\pmod p$, смотри об этом в тексте выдающегося современного ученого https://terrytao.wordpress.com/2009/08/18/the-least-quadratic-nonresidue-and-the-square-root-barrier/

[29] Но уже сейчас уместно отметить, что любые ссылки на этот алгоритм в ответах будут считаться допустимыми, только если вы умеете обосновывать его корректность.

[30] Следует отметить, что если изменить правила пересчета вероятности на те, которые (экспериментально) выполняются в микромире, и рассмотреть т.н. квантовые алгоритмы, то задачу факто-

решето и придумать нельзя, хотя, конечно, во всяком переборном алгоритме крайне важны константы; прочитать об этих процедурах можно, например, **[Кормен 1, §33.9]**. Задача факторизации имеет многочисленные криптографические приложения, где фактически используется в качестве примитива, одно из которых мы сейчас и рассмотрим.

Начнем с модельного примера выбора секретного ключа с использованием публичных каналов обмена информацией. Могут ли Алиса и Боб[31] договориться, например, *по телефону (который может прослушиваться)* о некотором *секретном ключе*, который они будут использовать в дальнейшем?

Сначала рассмотрим следующую **СХЕМУ 1**.

(*i*) Алиса и Боб выбирают большое простое число p и некоторое $1 < g < p$. Эта информация публичная и может быть перехвачена "нехорошим человеком" Евой.

(*ii*) Затем Алиса *секретно* выбирает число n, а Боб *секретно* выбирает число m.

(*iii*) Затем Алиса открыто передает Бобу число $A = ng \pmod p$, а Боб открыто сообщает Алисе число $B = mg \pmod p$.

(*iv*) Теперь оба могут легко вычислить "секретный ключ" $s = nmg \pmod p$.

Легко убедиться, что можно быстро (за полиномиальное время) найти s, зная p, g, A, B. В криптографических терминах это значит, что **СХЕМА 1** является ненадежной[32].

Рассмотрим также **СХЕМУ 2** выбора секретного ключа или схему обмена ключами Дифи и Хеллмана.[33] Хронологически эта схема предшествовала (и являлась мотивацией для) схемы RSA (см. ниже). Она очень похожа на первый вариант.

(*i*) Алиса и Боб выбирают большое простое число p и g — некоторый первообразный (примитивный) корень[34] $\pmod p$. Эта информация публичная и может быть перехвачена "нехорошим человеком" Евой.

(*ii*) Затем Алиса *секретно* выбирает число n, а Боб *секретно* выбирает число m.

(*iii*) Затем Алиса открыто передает Бобу число $g^n \pmod p$, а Боб открыто сообщает Алисе число $g^m \pmod p$.

(*iv*) Теперь оба могут легко вычислить "секретный ключ" $s = g^{mn} = (g^n)^m = (g^m)^n \pmod p$.

В настоящее время СХЕМА 2 считается надежной, поскольку Еве для дешифровки ключа предположительно нужно уметь быстро вычислять *дискретный логарифм*, т.е. решать уравнение $g^x = a \pmod p$, чего пока никто не умеет[35].

Но если Ева может не только подслушивать, но и выступать активным агентом, то дела Алисы и Боба осложняются. Например, если Ева может перехватывать и подменять сообщения, то она может послать Бобу *от имени Алисы* некоторое $g^t \pmod p$ и получить секретный ключ $g^{tm} \pmod p$ для дешифровки сообщений Боба. Такую же операцию Ева может провести и в отношении Алисы. Таким образом, возникает проблема идентификации участников. Эта задача решается, например, в очень распространенной схеме шифрования с *открытым ключом* **RSA**[36]. Состоит она в следующем.

(*i*) Боб выбирает *модуль*– число $n = pq$, равное произведению двух больших простых чисел.

(*ii*) Потом Боб выбирает *секретный ключ*– d (он известен только ему).

(*iii*) Затем Боб вычисляет *открытый ключ* $e = d^{-1} \pmod{(p-1)(q-1)}$.

(*vi*) Информация о (e, n)— публичная (например, Боб помещает ее в сеть).

(*v*) Если Алиса хочет послать секретное сообщение x Бобу, то она проводит шифровку (e{ncrypts}) $x \to e(x) = x^e \pmod n$ и посылает $e(x)$ *по открытому каналу*.

(*vi*) Боб легко дешифрует d{ecrypts}) сообщение с помощью секретного ключа $d(e(x)) \to (e(x))^d \pmod n = x \pmod n$.

Считается, что "нехороший человек" Ева не сможет прочитать сообщение, поскольку для этого ей нужно найти делители n.

Схема RSA позволяет также создавать защищенные электронные подписи. Пусть открытый ключ Боба $(e, n$. Если он хочет электронно "подписать" свое сообщение A, то должен послать сообщение $B = A^{\text{секр. ключ Боба}} \pmod n$ (для того чтобы идентифицировать "подпись" Боба его сообщение нужно преобразовать $B^e \pmod n$).

Резюме. *RSA — это асимметричная схема (для шифрования и дешифрования применяются разные процедуры), которая характеризуется следующими параметрами: $n = pq$, где p, q — различные большие простые числа; открытый ключ (e, n), где e взаимно просто с $\varphi(n) = (p-1)(q-1)$; секретный ключ (d, n), где d обратно к e по модулю $\varphi(n)$. Пусть M — остаток по модулю n. Тогда процедура шифрования сообщения M выглядит так: $P(M) = M^e \mod n$, а процедура дешифрования сообщения C выглядит так: $S(C) = C^d \mod n$. Криптоустойчивость схемы основана на предполагаемой сложности задачи* **факторизации** *(число n всем известно, но не понятно, как по нему вычислить $\varphi(n)$, если нам не известно разложение n на множители).*

Задача 27. (0.01+0.02). (*i*) Пусть открытый ключ Боба $(25, 2021)$. Он хочет послать сообщение (число) за своей подписью. В какую степень он должен его возвести?
(*ii*) Докажите или опровергните, что кодирование в системе RSA $M \to M^e \mod n$ биективно отображает отрезок $\{0, \dots, n-1\}$ в себя.

Краткий конспект
Показатели. Первообразные корни

Рекомендуем почитать **[Виноградов, глава 6]**.

Абелева мультипликативная группа Z_n^* (ненулевых) кольца вычетов по модулю n иногда бывает циклической. В этом случае любая ее образующая называется первообразным (примитивным) корнем. Следующая теорема, приписываемая К.Ф.Гауссу, дает ответ на вопрос, когда первообразные корни существуют.

Теорема 2 *В Z_n существует первообразный корень, если и только если $n = 2, 4, p^k, 2p^k$, $k = 1, 2, \dots$, p — нечетное простое число.*

Доказательство можно восстановить, если решить дополнительные задачи. В **[Кормене]** это утверждение не доказывается.

Напомним определения. Если a взаимно просто с n, то существуют положительные числа γ, для которых верно равенство $a^\gamma = 1 \pmod n$. Наименьшее из них называется **показателем a по модулю n**.

Из малой теоремы Ферма[37] $a^{\varphi(n)} = 1 \pmod n$ следует, что показатель всегда является делителем $\varphi(n)$.

ризации удается быстро решить. Рекомендуем прочитать об этом в книге **[К-Ш-В]**, хотя бы для того, чтобы правильно реагировать на многочисленные досужие рассуждения на эту тему, которые периодически появляются даже в таблоидах.

[31] Это стандартные персонажи криптографических протоколов. Им обычно противостоит "нехороший человек" Ева.

[32] На самом деле, свойство криптографической ненадежности гораздо слабее, чем существование полиномиального алгоритма, поскольку Алису и Боба также не устроит результат, когда Ева может достаточно **часто** дешифровывать сообщения. Обсуждению этих вопросов посвящена обширная литература.

[33] Предложена в статье (Diffie Whitfield; Hellman Martin E. New directions in cryptography. IEEE Trans. Information Theory IT-22 (1976), N 6, 644–654), где было определено само понятие криптографии с открытым ключом.

[34] Что это такое? Почему g существует, и как его находить? Сейчас просто вспомните определение, а ниже мы поговорим об этом более подробно.

[35] Хотя к утверждению о надежности нужно относится с известной долей скепсиса. Во всяком случае на сайтах, посвященных криптографии, буквально заклинают не использовать схемы "из книжек", поскольку они довольно успешно взламываются.

Кроме того, обратите внимание, что здесь мы, как и многие авторы криптографических текстов, слегка передернули. Формально перед Евой стоит не задача вычисления дискретного логарифма, а проблема вычисления ключа по известным g^n и g^m, а эта задача может оказаться проще, чем вычисление дискретного логарифма.

[36] Названной в честь авторов R{ivest}- S{hamir}- A{dleman}. По непроверенным данным RSA прочно удерживает первое место в мире по числу проданных патентов.

[37] $\varphi(\cdot)$ — функция Эйлера.

Если в Z_n есть первообразный корень g, то для чисел a, взаимно-простых с n, можно ввести понятие *индекса* или *дискретного логарифма*, в котором первоообразный корень играет роль основания логарифма. По определению, если $g^\gamma = a$, то число γ называтся *индексом вычета a по* $\pmod n$ *при основании g и обозначается* $ind_g a$ *или просто* $ind\, a$.

Теорема 3 *Для всякого простого модуля p существует ровно $\varphi(p-1) = \varphi(\varphi(p))$ первообразных корней, несравнимых по* $\pmod p$.

Доказательство. Пусть δ — какой-нибудь делитель $p-1$. Тогда если существует хотя бы один вычет a с показателем δ, то существует ровно $\varphi(\delta)$ несравнимых по $\pmod p$ вычетов с этим показателем. В самом деле, по определению, все вычеты с показателем δ удовлетворяют сравнению $x^\delta = 1 \pmod p$. Но все решения этого уравнения исчерпываются списком $1, a, a^2, \ldots, a^{\delta-1}$, поскольку все эти вычеты несравнимы между собой и удовлетворяют уравнению $((a^i)^\delta = (a^\delta)^i = 1 \pmod p)$, а больше, чем δ, решений быть не может (почему?). Осталось заметить, что вычет a^s имеет показатель δ, если и только если $\mathrm{НОД}(s, \delta) = 1$, и значит в списке есть ровно $\varphi(\delta)$ вычетов с показателем δ.

Обозначим $\psi(\delta)$ — число несравнимых по $\pmod p$ с показателем δ. Мы выяснили, что если $\psi(\delta) \neq 0$, то $\psi(\delta) = \varphi(\delta)$.

Теперь разобьем вычеты $0, 1, \ldots, p-1$ на группы, относя к одной группе все вычеты с одинаковым показателем. По построению, поскольку каждый вычет входит в какую-то группу, получим: $\sigma_{\delta|p-1}\psi(\delta) = p-1$. Но аналогичное равенство справедливо и для функции Эйлера $\forall n$, $\sigma_{d|n}\varphi(d) = \varphi(n)$. Следовательно, $\psi(\delta) = \varphi(\delta)$, в частности число первообразных корней по $\pmod p$ равно $\psi(p-1) = \varphi(p-1)$. □

Если в кольце Z_n есть первообразные корни, то рассуждения, аналогичные предыдущим, показывают, что показатель δ вычета a, взаимно простого с модулем n, определяется равенством $\mathrm{НОД}(ind\, a, \varphi(n)) = \frac{\varphi(n)}{\delta}$. В частности, число первообразных корней по $\pmod n$ (если они существуют) равно $\varphi(\varphi(n))$. Таким образом, первообразных корней иногда может быть "довольно много", и если мы можем быстро проверить является ли данный вычет первообразным корнем (например, если известны множители числа $\varphi(n)$), то можно искать первообразные корни, выбирая случайные числа, как и в алгоритме проверки простоты.

Как задача нахождения первообразного корня, так и вычисление дискретного логарифма считаются вычислительно тяжелыми задачами.

Задача 28. (0.01) Пусть вычет a имеет показатель δ_1 по $\pmod{n_1}$ и показатель δ_2 по $\pmod{n_2}$, причем модули n_1 и n_2 взаимно просты. Найдите показатель a по $\pmod{n_1 n_2}$.

Задача 29. (3×0.01) (*i*) Найдите все решения уравнения $\varphi(n) = 6$.
(*ii*) Найдите распределение вычетов по показателям Z_{19};
(*iii*) Найдите все первообразные корни $\pmod{19}$.

Задача Д–13. ($0.01 + 0.02 + 0.02$) (*i*) Найдите все первообразные корни в Z_2 и Z_4.
(*ii*) Покажите, что $\mathrm{Z}^*_{2^k} \cong \mathrm{Z}_2 \times \mathrm{Z}_{2^{k-2}}$ для $k > 2$, и поэтому порядок любого элемента в группе $\mathrm{Z}^*_{2^k}$, $k > 2$ не больше $\frac{1}{2}\varphi(2^k)$ (поэтому в кольцах $\mathrm{Z}_8, \mathrm{Z}_{16}, \ldots$ первообразных корней нет).
(*iii*) Пусть нечетный модуль n имеет два или более различных простых сомножителя, т.е. $n = n_1 n_2$, причем n — нечетное и $\mathrm{НОД}(n_1, n_2) = 1$. Покажите, используя задачу **№ 30**, что в Z_n первообразных корней нет. Аналогично проверяется, что первообразных корней нет по четному модулю $2^k n$, где нечетное число n имеет не менее двух различных простых делителей.

Замечание. Для доказательства Теоремы 2 осталось убедиться в существовании первообразных корней по модулям p^k и $2p^k$ $k > 1$, p — нечетное простое число.

Курсовая контрольная по материалам первой половины курса в понедельник 20.03 13:55–17:05 в Акт. зале и Б.хим.