АЛГОРИТМЫ И МОДЕЛИ ВЫЧИСЛЕНИЙ

Протокол

Вас ждут три курсовые контрольные. Они будут проходить по воскресеньям, соответственно, 24.03 (midterm), 12.05 (final) и 19.05 (утешительная) в 110 и 115 КПМ. Начало 9:00. Ориентировочное время: 3 часа. Мы постараемся проверять и показывать работы в тот же день и в тех же аудиториях.

Как и в прошлом году, будет организована поддержка курса в системе piazza, хотя, надо признать, она использовалась только в бухгалтерских целях, а ее основная ее функция, налаживание конструктивного диалога между студентами и преподавателями, практически востребована не была.

Тут, безусловно, нужна обратная связь. В нулевом приближении будут разбираться задачи из текущего недельного задания, которые вызвали трудности, и/или связанные темы. Информация о темах следующего занятия будет размещаться в piazza.

Обязанности сторон и правила аттестации Пожалуйста, отнеситесь к ним серьёзно Кроме прав в них указаны также и обязанности!

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 завершиться в течение четырех недель, т. е. до 4.03.2019.

2. Протокол получения оценки следующий

(*i*) Студент, получивший положительную оценку по **обеим курсовым контрольным** и "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 **[округление вниз]**, 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 (при этом оценка может быть как повышена, так и понижена).

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

(*ii*)-а. Если утешительная контрольная написана на положительную оценку, то вопрос о зачёте решается **в результате последующего устного опроса при показе работы**. При этом, 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 зачётной оценки нужно следить за графиком пересдач (и процесс продолжается). На пересдачах проводятся **короткие письменные контрольные и обязательный устный опрос**.

3. Идущее ниже задание является **официальным**. В частности, это означает, что при возникновении каких-то коллизий (например, при незачётах и/или стремлении повысить оценку, полученную по протоколу и т.д.) преподаватели 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: понедельный план

Приводим понедельный план наших занятий. Хотим подчеркнуть, что для студентов, которые привыкли к регулярным занятиям, никаких особых трудностей в курсе нет. Но они точно могут возникнуть, если вы собираетесь придерживаться стандартной методики: начинать изучение за пять минут до (или после) начала зачёта. И причина этого та же, что и была на ТРЯП: проблемы возникают с изучением нескольких нетривиальных понятий, к которым, среди прочего, надо просто привыкнуть, и на что может уйти некоторое время, скажем, неделя или месяц, даже при регулярном продумывании.

1. **Темы 1-й недели 4.02-10.02. Повторение. 1 лекция.** Формальное определение алгоритма. Емкостная и временная сложность алгоритма. Различные определения сложности алгоритма. Эффективные (полиномиальные алгоритмы). Примеры алгоритмов и иллюстрация принципов разработки алгоритмов: алгоритм Евклида, проверка простоты и факторизация чисел; Проверка принадлежности слова регулярным и КС-языкам. Непустота пересечения регулярных и КС-языков.

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. Решение линейных рекуррентных уравнений *и линейных разностных матричных уравнений.*

2. **Темы 2-й недели 11.02-17.02. 2-я лекция.** Теорема о линейном ускорении. Теоремы о временной и емкостной иерархии (без доказательства). Определения сложностных классов ($\mathcal{P}, \mathcal{NP}, \mathcal{PSPACE}$).

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.

3. **Темы 3-й недели 18.02-24.02. 3-я лекция.**
ВНИМАНИЕ! ПЕРЕНОС ЗАНЯТИЙ! Лекция и семинар перенесены с субботы 23.02 на понедельник 25.02: лекция пройдет в Б.Физ. 10:45–12:10; семинар у моих студентов пройдёт в 428 ГК 12:20–13:45.

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.

4. **Темы 4 недели 25.02-3.03 4 лекции.** Класс $\mathcal{PSPACE}$. Теорема Савича $\mathcal{PSPACE} = \mathcal{NPSPACE}$ (без доказательства). Полные языки в $\mathcal{PSPACE}$: истинность булевской формулы с кванторами; эквивалентность конечных автоматов.

5. **Темы 5-й недели 4.03-10.03. 5-я лекция.** Элементарные вероятностные понятия. Мотивировка: "наивная" точка зрения: вероятность — это отношения (доля, частота) числа благоприятных исходов к общему числу исходов. Вероятностное пространство. Случайные величины. Математическое ожидание и его свойства. Дисперсия. условные вероятности. Независимость. Теорема Байеса. Закон больших чисел. Оценка хвостов. Неравенства Маркова и Чебышева.

Большие уклонения, неравенство Чернова.

6. **Темы 6-й недели 11.03-17.03. 6-я лекция.** Вероятностные алгоритмы. Классы $\mathcal{RP}$, $\mathcal{BPP}$, $\mathcal{ZPP}$. Проверка полиномиальных тождеств; поиск паросочетаний в графах; алгоритм Каргера поиска минимального разреза. Приближённые вероятностные алгоритмы поиска максимального разреза. и его дерандомизация. Лемма Шварца–Зиппеля. Дерандомизация.

7. **Темы 7-й недели 18.03-24.03. 7 лекция.** Числовые алгоритмы[1]. 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. Вероятностные алгоритмы проверки простоты.

В воскресенье 24.03 с 9:00–12:00 в 110 КПМ и 115 КПМ пройдет первая курсовая контрольная работа по темам 1–5. Мы постараемся проверить и показать её в тот же день и

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

в тех же аудиториях. Оценки за обе контрольные серьёзно влияют на получение зачёта по курсу.

Если не будет специально оговорено, то предполагаемый формат регулярных контрольных open-book, no device. Обычно разрешается взять лист формата A4 с произвольными записями.

Ещё раз повторим: по протоколу студенты, получившие неудовлетворительную оценку на первой или второй курсовых контрольных, не могут быть аттестованы без получения положительных оценок на последующих контрольных (т. е. они обязаны написать утешительную контрольную). Это же замечание относится и к студентам, пропустившим контрольную *даже по уважительной причине.*

8. **Темы 8-й недели 25.03-31.03 8-я лекция.** Хеш-таблицы. Разрешение коллизий с помощью цепочек. Хеш-функции (деление с остатком, умножение). Универсальные и k-универсальные хеш-функции.

Структуры данных. Стек и очередь. Двоичная куча (пирамида). Двоичное дерево поиска и его сбалансированные вариации. Амортизационный анализ.

9. **Темы 9-й недели 1.04-7.04. 9-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.

10. **Темы 10-й – недели 8.04-14.04. 10-я лекция.** Алгоритмы сортировки: "пузырёк, "быстрая сортировка (quicksort); сортировка с помощью кучи (heapsort); сортировка слиянием (mergesort). Анализ трудоёмкости алгоритма quicksort по наихудшему случаю и в среднем. Устойчивость алгоритма сортировки. Цифровая сортировка. Нижние оценки в модели разрешающих деревьев.

11. **Темы 11-й недели 15.04-21.04. 11-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. Процедура слияния множеств.

12–13. **Темы 12-й недели и 13-й недели 22.04-5.05. 12–13-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. Лемма изоляции (isolation lemma). Полиномиальные алгоритмы ЛП.

14. **Тема 14-й недели 6.05-12.05. 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.

В воскресенье 12.05 с 9:00–12:00 в 110 КПМ и 115 КПМ пройдет вторая курсовая контрольная работа по всем темам курса. Оценки за обе контрольные серьёзно влияют на получение зачета по курсу.

Если не будет специально оговорено, то предполагаемый формат регулярных контрольных open-book, no device. Обычно разрешается взять лист формата A4 с произвольными записями, а также одну или две книги (обычно это Кормен и Дасгупта).

Через неделю 19.05 с 9:00–12:00 в 110 КПМ и 115 КПМ пройдёт утешительная контрольная работа по всем темам курса. Её формат будут уточнён. Она обязательна для всех студентов, которые не будут аттестованы к моменту её проведения.

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. **[К-Ш-В]** *Китаев А., Шень А., Вялый М.* Классические и квантовые вычисления. М.: МНЦМО-ЧеРо, 1999.
5. **[Кормен 1]** *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: Построение и Анализ. М.: МЦНМО, 2002.
6. **[Кормен 2]** *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы: Построение и Анализ. (2-е изд.) М.: Вильямс, 2005.
7. **[Кормен 3]** *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы: Построение и Анализ. (3-е изд.) М.: Вильямс, 2013.
8. **[ХМУ]** *Хопкрофт Д., Мотвани Р., Ульман Д.* Введение в теорию автоматов, языков и вычислений. М.,: Вильямс, 2002.
9. **[AB]** *Arora S., Barak B.* Computational Complexity: A Modern Approach. theory.cs.princeton.edu/complexity/book.pdf
10. **[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. **[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. Затем было выпущено издательством "Вильямс". Есть многочисленные сетевые варианты.
5. **[К-Ф]** *Кузюрин Н., Фомин С.* Эффективные алгоритмы и сложность вычислений. М.: МФТИ, 2007.
6. **[П-С]** *Пападимитриу Х., Стайглитц К.* Комбинаторная оптимизация. Алгоритмы исложность. М.: Мир, 1985.
7. **[Феллер]** *Феллер В.* Введение в теорию вероятностей и ее приложение. Т. 1. М.: Мир, 1984.
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)

Задание состоит из списка недельных заданий (указаны даты обсуждения на семинарах и/или сдачи соответствующих задач). Каждое недельное задание состоит из нескольких 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.
Длина слова x обозначается $|x|$.

Когда это необходимо выделить, *унарная* и *двоичная (бинарная)* запись числа "x" обозначаются x_{unary} и x_{binary}, соответственно.

Производящей функцией последовательности $\{a_n, n = 0, 1, \dots\}$, по определению, называется *формальный* степенной ряд $\sum_{i=0}^{\infty} a_n t^n$. Термин "формальный" означает, что это просто **другой способ записи последовательности**. С производящими функциями можно проводить арифметические операции, а *если им удаётся придать аналитический вид, т. е. считать, что t — это вещественная или комплексная переменная и соответствующий ряд сходится*, то с ними можно обращаться как с обычными функциями, что иногда позволяет эффективно производить вычисления.

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 $\gamma = 0.57721\dots$ — это так называемая константа Эйлера. Кроме того, нужно знать, определение чисел Каталана $c_n = \frac{1}{n+1}\binom{2n}{n}$ (c_n имеет множество комбинаторных интерпретаций, например, равно числу правильно построенных скобочных выражений или путей Дика длины $2n$) и их производящая функция равна $D(t) \stackrel{def}{=} \sum_{n=0}^{\infty} c_n x^n = \frac{1-\sqrt{1-4x}}{2x}$.

Мы также будем использовать элементарные понятия теории вероятностей. При этом вероятность будет пониматься "наивно" как отношение числа благоприятных исходов к общему числу исходов.

При работе с величинами, чьи значения зависят от случайных исходов мы будем использовать понятие **математического ожидания**, которое, по определению, равно средневзвешенному значению величины по всем исходам. Математическое ожидание величины ξ обозначается $\mathbb{E}\xi$.

Пример. Пусть ν равно числу бросаний симметричной[2] *монетки до первого выпадения герба. Тогда $E\nu = \sum_{i=1}^{\infty} i \times (\frac{1}{2})^i = 2$, поскольку вероятность выпадения первого герба на i-м шаге равна $(\frac{1}{2})^i$.*

Очень важным свойством математического ожидания является его линейность, что иногда позволяет значительно упрощать вычисления.

Случайность может рассматриваться как важный (а, может быть, и важнейший) алгоритмический ресурс и определённая часть курса посвящена расшифровке этого тезиса.

Задание на первую неделю 4.02–10.02 [0.08]
Оценки. Рекуррентные последовательности.
Литература: [Кормен 1, Глава 1]
[Кормен 3, Гл. 3–5], [ДПВ. Гл. 2], [Феллер, Гл. 3]

`https://en.wikipedia.org/wiki/Matrix_difference_equation`

После номера задачи в круглых скобках стоит её стоимость.

Задача 1. (2×0.02). Пожалуй, самый известный алгоритм,— это *алгоритм Евклида*:

Input: Даны целые неотрицательные числа $x > y \in \mathbb{N}$
Output: $\gcd(x, y)$

```
1 if y = 0 then return x;
2 if y = 1 then return 1;
3 return gcd(y, x mod y)
```

Algorithm 1: Алгоритм Евклида

Поскольку на каждой итерации по крайней мере одно число уменьшается, то вопрос о конечности процедуры не возникает. Более того, ясно, что нам понадобится не более, чем $O(|x_{unary}|)$ итераций. В частности, если $x = 2^{200}$, то оценка превышает число протонов во вселенной, т. е. практически бессмысленна. Вот, если бы удалось получить оценку вида $O([|x_{binary}|]^{O(1)})$, как говорят "полиномиальную" (по длине [*двоичной*] записи), то это было бы гораздо более убедительным свидетельством эффективности алгоритма. Хотя, как обычно, ко всем таким декларациям нужно относиться аккуратно, поскольку важен не только качественный вид асимптотики, но и конкретные константы в записи отношений порядка.

Лозунг. Основной задачей курса является разработка полиномиальных алгоритмов и изучение препятствий к их существованию для конкретных задач.

Попробуем получить более точную оценку трудоемкости. Пусть для $1 \le i \le m$ x_i и, соответственно, y_i обозначают значения параметров x и y на i-й итерации алгоритма (например, $x_1 = x, y_1 = y$). Также положим $s_i \stackrel{def}{=} x_i + y_i$.

(*i*) Покажите, что $1 \le i \le m$: $s_i \le 2/3 \cdot s_{i-1}$.

(*ii*) Вычислите $\gcd(F_{m+2}, F_{m+1})$, где F_n — это n-е число Фибоначчи.

Замечание 1 *Из (i) вытекает, что число итераций $m \le \log_{1.5}(x+y)+1$. В 1844 году Габриэль Ламе показал, что если алгоритм Евклида требует m шагов для пары чисел (x, y), $x > y$, то выполняются следующие неравенства $x \ge F_{m+2}$ и $y \ge F_{n+1}$ (доказать это можно по индукции). Отсюда получаем чуть более точную оценку:*

$$x + y \ge F_{m+2} + F_{m+1} = F_{m+3} \gtrapprox \phi^{m+1},$$

так что $m \lessapprox \log_\phi(x+y)$, где $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803398875$ — это золотое сечение.

Задача 2. (0.02). Вычислите число правильно составленных скобочных выражений, содержащих $2n$ скобок в которых в любом непустом префиксе число открывающих скобок больше числа закрывающих.

Подсказка. *Это стандартная задача о числе стартующих из начала координат путей по целым точкам шагами $(1, 1)$ и $(1, -1)$, которые далее ось $0X$ не пересекают. Эти пути иногда называют* **путями Дика**.

Задача 3. (3×0.02). Имеются окрашенные прямоугольные таблички трёх видов: чёрный квадрат размером 2×2, белый квадрат размером 2×2 и серый прямоугольник размером 2×1 (такую табличку можно поворачивать на 90^0). Нужно подсчитать число различных способов Φ_n, которыми можно замостить полосу размера $2 \times n$.

Считается, что полоса должна быть заполнена табличками без пробелов и без наложений (касания возможны только по границам). Два способа замощения считаются одинаковыми, если они совпадают при наложении (параллельном переносе) полосок.

(*i*) Найдите Θ– асимптотику Φ_n.

(*ii*) Найдите явную аналитическую формулу для Φ_n.

(*iii*) Объясните, как корректно определить Φ_0.

Подсказка. Составьте рекурсию и решите её.

Задача 4. (0.02). Найдите Θ-асимптотику рекуррентности, которая определяется в следующем тексте.

Colour the edges of a complete graph of n vertices by three colours so that the number of tirangles all whose edges get a different colour is maximal. Denote this maximum by $G_3(k)$. They conjectured that $G_3(k)$ is obtained as follows: clearly

[2] У такой монетки вероятности выпадения герба и решётки одинаковы.

$G_3(1) = G_3(2) = 0, G_3(3) = 1, G_3(4) = 4$. Suppose $G_3(k_1)$ has already been determined for every $k_1 < k$. Then

$$G_3(k) = G_3(u_1) + G_3(u_2) + G_3(u_3) + G_3(u_4) + \\ + u_1u_2u_3 + u_1u_2u_4 + u_1u_3u_4 + u_2u_3u_4,$$

where $u_1 + u_2 + u_3 + u_4 = k$ and the u's are as nearly equal as possible.

Замечание 2 *Гипотеза о том, что точное решении рекуррентности является ответом указанной экстремальной комбинаторной задачи открыта. В 2014 году удалось показать, что $G_3(\cdot)$ действительно удовлетворяет рекуррентности для достаточно больших n и также для всех $n = 4^k$. Детально пересказать аргументы достаточно проблематично, поскольку доказательство существенно опирается на (компьютерный) вывод в некоторой формальной системе.*

Подсказка. Нахождение асимптотики — простая задача. Подумайте, как обосновать округление, которое придётся проводить.

Задача Д–1. (0.02). Назовём правильное скобочное выражение *блоком*, если нем первая (открывающая) и последняя (закрывающая) скобки являются парными. Вычислите число $g_k(n)$ правильных скобочных выражений, имеющих $2n$ скобок, которые состоят ровно из $k \geq 1$ блоков.

Подсказка. Это в точности задача о вероятности k возвратов в начало при случайном блуждании на прямой с "отражением" в нуле (всегда уходим из нуля вправо).

Задача 5. (3×0.02). Пусть A_n — число натуральных решений уравнения $2x + 3y = n$, т. е. $A_1 = A_2 = A_3 = A_4 = 0$, $A_5 = 1\,(x = 1, y = 1), \ldots$.
(i) Найдите производящую функцию последовательности A_n, $n = 1, 2, \ldots$.
(ii) Найдите θ-асимптотику A_n.
(iii) Найдите явное аналитическое выражение A_n.

Подсказка. Убедитесь, что искомая производящая функция вычисляется явно.

Задача Д–2. (3×0.02). Построим последовательность полиномов $U_n(x) = xU_{n-1}(x) + U_{n-2}(x)$; $U_0(x) = 0$; $U_1(x) = 1$.
(i) Покажите, что максимальное число итераций алгоритма Евклида (в кольце $\mathbb{Q}[x]$) на полиномах степени n достигается на паре $\gcd(U_n(x), U_{n-1}(x))$.
(ii) Покажите, что полином $U_n(x)$ неприводим (над $\mathbb{Q}$) тогда и только тогда, когда n простое.
(iii) Покажите, что $\gcd(U_a(x), U_b(x)) = U_{\gcd(a,b)}(x)$.

Быстрое умножение и возведение в степень
Краткий конспект
Литература: [Кормен 1, §31.2]
Литература: [Кормен 3, §4.2], [ДПВ, §2.5]

С этими материалами полезно ознакомиться. Часть из них, например, быстрое возведение в степень, будет использоваться в дальнейшем.

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 другой алгоритм умножения n-значных чисел A и B (алгоритм А.А. Карацубы[3] 1959 г.). 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: $AB = A_1B_1 + 2^{\frac{n}{2}}((A_1 + A_2)(B_1 + B_2) - (A_1B_1 + A_2B_2)) + 2^nA_2B_2$.

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 $\varphi(n) = O(n^{\log 23} = O(n^{1.5849\ldots})$.

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 $(A_1 + A_2)(B_1 + B_2) = a_1b_1 2^n + (a_1b_2 + a_2b_1)2^{\frac{n}{2}} + a_2b_2$. Член a_2b_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[4], т. е. разбить матрицу порядка $n \times n$ на четыре блока порядка $\frac{n}{2} \times \frac{n}{2}$ и провести перемножения блоков по формулам, записанным выше (**нужно обязательно обосновать корректность процедуры!**), то получится такая рекуррентная оценка трудоёмкости алгоритма перемножения матриц: $T(n) = 7T(\frac{n}{2}) + O(n^2)$.

[3] В некоторых источниках, например, даже в **[ДПВ]**, совершенно безосновательно утверждается, что подобный алгоритм использовал ещё К.-Ф. Гаусс, см. по этому поводу https://www.mccme.ru/free-books/matpros/pdf/МП24-Шень.pdf

[4] Подробности можно прочитать, например в **[Кормен 1, §31.2]**, а еще лучше самостоятельно обосновать рекурсивную процедуру.

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 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 эффективная процедура перемножения матриц с асимптотической трудоемкостью $O(n^{2+\varepsilon})$.** Такой алгоритм может (гипотетически) перевернуть весь мир вычислительной линейной алгебры. Замечу, что несколько лет назад в задаче поиска наилучшего алгоритма умножения матриц произошло первое за последние 30 лет продвижение. Кстати, как это ни покажется удивительным и даже неуместным в подобном контексте, связанная с этим результатом история драматична. Об этом можно почитать, например, http://www.scottaaronson.com/blog/?p=839#comments

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 идёт 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 её выполнения нужно порядка n умножений. Оценим сложность алгоритма. Представим себе, что умножение занимает один такт времени[5]. 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 трудоёмкость 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 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 чётности 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 идёт от старших битов к младшим).

Описание алгоритма следующее. В двоичной записи показателя n (нули слева убраны) произведём следующую замену символов $1 \to SA$ и $0 \to S$. В полученном слове удалим начальную пару символов SA. В результате получим "код программы" на следующем диалекте: $S[quare]$ 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.

Трудоёмкость обоих алгоритмов значительно меньше, чем в исходном "наивном алгоритме", и даже растёт 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 игрушечной постановке вовсе не праздный. Представьте себе, что вам нужно произвести не одно, а один гугл возведений в ту же степень n (при разных основаниях a), тогда экономия даже одного умножения может оказаться существенной.

Отметим, что индийский алгоритм неоптимальный. Например, a^{54} можно вычислить, 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[6].

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[7] для вычисления функции $l(n)$. Не решены также мно-

[5] 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 сомножителей*, а длина записи сомножителей растёт на каждой итерации примерно в два раза.

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

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

гие 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): $n = d_0 m^t + d_1 m^{t-1} + \ldots + d_t$. 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 соответствует 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 соответствует возведению a^{d_0} в m-ю степень и умножению на a^{d_1}. Далее рассмотрим цепочку $2(md_0 + d_1), 4(md_0 + d_1), \ldots, m(md_0 + d_1), m^2 d_0 + md_1 + d_2$ и т.д., пока не получим n. Для наглядности выпишем всю цепочку.

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

На практике, конечно, нужно вычеркнуть из неё 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 2-ю неделю 11.02-17.02. [0.12] Полиномиальные алгоритмы. Класс $\mathcal{P}$ Литература: [Кормен 1, Глава 36] [Кормен 3, Глава 34], [ДПВ. Глава 8]

Мы приступаем к исследованию формальных свойств алгоритмов, трудоёмкость которых является полиномиальной по длине входа. Такие алгоритмы мы считаем эффективными[8].

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 $\text{poly}(|x|)$, 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, $i = 1, 2, \ldots$ можно сопоставить её *график:* полиномиальный предикат: $L_T = \{x_1 * y_1, x_2 * y_2, \ldots\} \in \mathcal{P}$.

Дадим эквивалентное определение класса $\mathcal{P}$. Пусть $L \subseteq \Sigma^*$ — разрешимый язык. Как мы помним (например, из курса ТФС), это означает, что существует распознающая его Машина Тьюринга (МТ) T_L, которая принимает (т. е. останавливается в принимающем состоянии) все слова из L и отвергает (т. е. останавливается в отвергающем состоянии) все слова из дополнения $\Sigma^* \setminus L$. Полезно вспомнить, что сама по себе разрешимость не предполагает существования каких-то оценок на число тактов работы T_L на произвольном слове $w \in L$.

Скажем, что $L \in DTIME(f(\cdot))$, где $f : \mathbb{N} \to \mathbb{N}$, если для любого $w \in \Sigma^*$ число шагов, которые делает МТ T_L на входе w, не превышает $f(|w|)$. (Детерминированная) МТ T_L называется *полиномиальной*, если $L \in DTIME(p(\cdot))$, где $p(x) \in \mathbb{N}[x]$, а класс $\mathcal{P}$ состоит из объединения всех языков, распознаваемых полиномиальными МТ:

$$\mathcal{P} = \cup_{k \ge 0} DTIME(O(n^k))$$

.

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 приведёнными 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 начнём изучать методы, позволяющие заключить, что подобных алгоритмов [для этих конкретных задач] не существует.

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 изучения этой темы аналогичны тем, которые возникли, например, в прошлом году при изучении регулярных языков: в предыдущем опыте вы почти не сталкивались ни с такой тематикой, ни с аналогичными формальными или неформальными конструкциями, помогающих правильно сориентироваться. Это, прежде всего, означает, что стандартный метод обучения: начать изучение за три минуты до экзамена или зачёта, может оказаться крайне непродуктивным. К этой теме нужно просто "привыкнуть", что подразумевает постоянное продумывание возникающих вопросов.

Полезно вспомнить тему вычислимости, которую вы изучали совсем недавно на курсе ТФС, и понятие недетерминированного вычисления, которую мы разбирали в курсе ТРЯП на примере простейшего недетерминированного устройства НКА. Грубо говоря, теперь мы хотим понять, можно ли эффективно моделировать индетерминизм на общих вычислительных устройства типа МТ. Как мы помним, для автоматов переход от НКА к ДКА всегда возможен, но требует, вообще говоря, экспоненциального увеличения памяти (числа состояний автомата). Для МТ пока удовлетворительного ответа нет.

[8] 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 следует 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 увидели, несмотря на внешнюю простоту описания, может быть весьма хитрым и контраинтуитивным.

Несмотря на то, что все уже неоднократно встречались с алгоритмами, но, тем не менее, для того, чтобы быстрее понять особенности нашего курса, мы начнём с повторения части алгоритмов, которые вы знаете из ТРЯП. Более того, такая точка зрения даже предпочтительнее, поскольку, как было замечено в самом начале занятий, мы хотим рассматривать алгоритмы в формате распознавания языков.

Задача 6. (3×0.01 балла). (i) Языки $L_1, L_2, \ldots, L_{2019}$ заданы регулярными выражениями. Постройте полиномиальный алгоритм проверяющий, что их пересечение не пусто, т. е. $\cap_{i=1}^{2019} L_i \neq \emptyset$.
(ii) Следует ли из решения предыдущей задачи, что проверка непустоты пересечения конечного семейства ДКА (в фиксированным алфавите, например, унарном) принадлежит классу $\mathcal{P}$?
(iii) Является ли этот язык разрешимым?

Задача 7. (0.01 балла). Опишите любой известный вам полиномиальный алгоритм построения минимального автомата по ДКА. Вы обязаны доказать, что ваш алгоритм полиномиальный, и как можно точнее оценить его трудоёмкость.

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, причём описание каждого языка даётся 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[9]:
$L(\cdot) \stackrel{?}{=} \emptyset$ — 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–3. ($2 \times 0.01 + 3 \times 0.02$) Покажите, что в колонках 2–6 таблицы можно поставить знак $\mathcal{P}$.

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

[9] 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 ещё работает, существенно отличаются.

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]**.

Полиномиальность метода Гаусса.

Задача 8. ($0.01 + 0.02$) Рассмотрим систему линейных уравнений $Ax = b$ с целыми коэффициентами, имеющую m уравнений и n неизвестных, причём 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 по длине входа алгоритм, т. е. алгоритм, имеющий трудоёмкость $\mathrm{poly}(m, n, \log H)$. 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 $(7 \cdot 6 + 5 \cdot 10 = 92,\ 6 \cdot 10 = 50)$, определении $\mathrm{НОД}(92, 60) = 2$ 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 $D_{ij}^{(k)}$, $k+1 \leq i, j \leq n$— 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 $d_{ij}^{(k)} = d_1 \ldots d_k a_{ij}^{(k)}$ и $det(D^{(k)}) = d_1 \ldots d_k$. Таким образом, можно все время работать с дробями, числители и знаменатели которых *являются минорами* **исходной** *матрицы*, так что длина записи остаётся полиномиальной[10], а все вычисления по методу Гаусса (включая, конечно, вычисления НОД получаемых дробей) будут также полиномиальными.

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

Из решения предыдущей задачи вытекает, что язык $L_{det} = \{A \in \mathbb{Z}^{n \times n} \mid detA \neq 0\} \in \mathcal{P}$.

Задача Д–4. (0.04) Докажите, что язык $L_{det}^{\bmod m} = \{A \in \mathbb{Z}^{n \times n} \mid detA \neq 0 \pmod m\} \in \mathcal{P}$. Отдельно разберите случаи, когда m — простое и когда составное.

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

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 9. (0.02) Покажите, что класс $\mathcal{P}$ замкнут относительно $*$-операции Клини ($L^* = \varepsilon \cup L \cup L^2 \cup \ldots$).

Недетерминированные полиномиальные алгоритмы
Класс $\mathcal{NP}$
Литература: [Кормен 1, Глава 36] [Кормен 3, Глав 34], [ДПВ. Глава 8]

Класс $\mathcal{NP}$ составляют предикаты (= свойства, языки), которые можно **проверить** с помощью полиномиальных по входу алгоритмов. Иными словами, мы хотим понять, какие возможности можно получить, если добавить в алгоритм (МТ) индетерминизм, но сохранить ограниченный ресурс: полиномиальное время. На сегодняшний день этот вопрос является центральным в теории вычислений и даже проник в каком-то виде в среду практиков и даже масс-медиа.

Итак, 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[11] 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"[12]), 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).

По аналогии с детерминированными вычислениями можно ввести класс $L \in NTIME(f(\cdot))$, где $f : \mathbb{N} \to \mathbb{N}$, такой что для любого $w \in \Sigma^*$ число шагов, которые делает недетерминированная МТ (НМТ) T_L на входе w, не превышает $f(|w|)$. НМТ T_L называется *полиномиальной*, если $L \in NTIME(p(\cdot))$, где $p(x) \in \mathbb{N}[x]$, а класс $\mathcal{NP}$ состоит из объединения всех языков, распознаваемы полиномиальными НМТ:

$$\mathcal{NP} = \cup_{k\geq 0} NTIME(O(n^k))$$

.

Легко убедиться, что класс $\mathcal{NP}$ как множество языков замкнут относительно операций объединения, пересечения и конкатенации. Например, NP-характеризация пересечения языков задаётся следующим образом. Пусть $L_1 = \{x \mid \exists y_1\ |y_1| \leq p_1(|x|) \wedge R_1(x,y_1)\}$ и $L_2 = \{x \mid \exists y_2\ |y_2| \leq p_2(|x|) \wedge R_2(x,y_2)\}$. Тогда $L = L_1 \cap L_2 = \{x \mid \exists y \stackrel{def}{=} (y_1,y_2)\ |y| = |y_1| + |y_2| \leq p_1(|x|) + p_2(|x|) \wedge R(x,y) \stackrel{def}{=} [R_1(x,y_1) \wedge R_2(x,y_2)]\}$, здесь $p_1(\cdot), p_2(\cdot) \in \mathrm{poly}(x)$.

Задача 10. (0.01) Покажите, что класс $\mathcal{NP}$ замкнут относительно $*$-операции Клини.

Укажите, как построить для результирующего языка соответствующий сертификат "y" и проверочный предикат $R(x,y)$, и обратите внимание, насколько доказательство легче аналогичного утверждения для класса $\mathcal{P}$.

Обратите внимание, что в списке операций отсутствует операция дополнения, и класс $\mathcal{NP}$ может быть и *не замкнут относительно дополнения* (если $\mathcal{P} \neq \mathcal{NP}$, то это заведомо так, почему?). Дополнительный класс обозначается $co-\mathcal{NP} \stackrel{def}{=} \Sigma^* \setminus \mathcal{NP}$. Таким образом, в $co-\mathcal{NP}$ входят все языки, дополнения которых принадлежат $\mathcal{NP}$.

Задача 11. (3×0.01) (i) Приведите пример какого-нибудь языка из $co-\mathcal{NP}$.
(ii) Выразите $co-\mathcal{NP}$ по аналогии с $\mathcal{NP}$ через сертификаты и проверочные предикаты.
(iii) Опишите словами язык, дополнительный к рассмотренному выше языку гамильтоновых графов и предложите какой-нибудь алгоритм распознавания этого языка.

В следующей задаче нужно *непосредственно использовать* определение сложностного класса $\mathcal{NP}$ и указывать соответствующие сертификаты y и проверочные предикаты $R(x,y)$.

Задача 12. (2×0.02) (i) Докажите, что язык $L_{det}^{\mathrm{mod}\ m} = \{A \in \mathbb{Z}^{n\times n} \mid detA \neq 0 \pmod m\} \in \mathcal{NP}$. Отдельно разберите случаи, когда m — простое и когда составное.
(ii) Покажите, что классу $\mathcal{NP}$ принадлежит язык $L_{cons} = \{(A \in \mathbb{Z}^{n\times m}, b \in \mathbb{Z}^n) \mid \exists x \in \mathbb{R}^m : Ax = b\}$ совместных систем линейных уравнений с целыми коэффициентами. Числа представлены своей двоичной записью.

Задача Д–5. (4×0.02). Пусть задана система линейных неравенств с целыми коэффициентами и мы хотим проверить, что у неё нет ни одного целочисленного решения. Ограничимся случаями, когда переменных две или три. Обозначим L_2 и, соответственно, L_3, языки всех таких систем. Докажите следующие утверждения.
(i) $L_2 \in \mathcal{NP}$;
(ii) $L_3 \in \mathcal{NP}$;
(iii) $L_2 \in \mathcal{P}$;
(iv) $L_3 \in \mathcal{P}$.

Ясно, что утверждения не независимы, и из пункта (iv) вытекают все предыдущие. Может показаться странным, но решение этого пункта было найдено только в 1980 году, хотя он может быть сформулирован совсем по-школьному: в $\mathbb{R}^3$ задан рациональный (т.

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

[12] Предложил А.Шень.

е. имеющий рациональные вершины) тетраэдр T, причём числители и знаменатели компонент вершин не превышают по модулю h. Можно ли, используя $\mathrm{poly}(\log h)$ операций, проверить, что в T нет целых векторов, т. е. $T \cap \mathbb{Z}^3 = \varnothing$?

Простые числа имеют короткие сертификаты

Приведём 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 $B = \mathbb{N} \setminus A = \{2, 3, 5, 7, 11, \ldots\}$ простых чисел[13]. 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 ещё умеем[14], 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 $g^{p-1} = 1 \pmod p$, $g^{\frac{p-1}{p_1}} \neq 1 \pmod p, \ldots, g^{\frac{p-1}{p_k}} \neq 1 \pmod p$, 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 $\log p$. К этой задаче мы ещё вернёмся.

Задача 13. (0.02) Постройте NP-сертификат простоты для числа $p = 2687$. Известно, что первообразный корень есть на отрезке $[2, 10]$. Простыми в рекурсивном построении считаются только числа $2, 3, 5$ (они сами являются своими сертификатами).

Задание на третью неделю: 18.02- 24.02 [0.05]
Классы $\mathcal{NP}$ и $co-\mathcal{NP}$
Полиномиальная сводимость. NP-полные языки
Литература: [Кормен 1, Глава 36] [Кормен 3, Глава 34], [ДПВ, Глава 8]

Определим одно из важнейших понятий курса.

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 действительно совершенно пустяковый вопрос[15]: верно 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 проверить принадлежность произвольного слова $w \overset{?}{\in} \mathcal{A}$ 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 $w \overset{?}{\in} \mathcal{A}$ 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 $\mathcal{A}$. Две следующие задачи иллюстрируют эти соображения.

Задача 14. (0.02) Докажите, что если можно обратить $n \times n$ матрицу за время $T(n)$ и $T(3n) = O(T(n))$, то существует $O(T(n))$-алгоритм умножения $n \times n$ матриц (использующий процедуру обращения в качестве подпрограммы). **Иначе говоря, умножение матриц не сложнее обращения матриц.**

Немного сложнее технически доказать и обратное утверждение: **обращение матриц не сложнее умножения матриц**, т. е. эти задачи не просто сводятся друг к другу, но и вообще *можно построить такие сводимости, что задачи имеют одинаковую сложность с точностью до умножения на константу!*

Задача построения выпуклой оболочки состоит в следующем. Даны 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 $V \subseteq S_V$.

Задача Д–6. (0.02) Рассмотрим модель вычислений, в которой за единицу времени можно выполнять операции сравнения, сложения и умножения возведения чисел в квадрат (т. е. вычисление x^2 по заданному x). Докажите, что в описанной модели вычислений задача сортировки n чисел за **линейное** время сводится к задаче построения выпуклой оболочки[16] 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 на плоскости, поскольку алгоритмы, имеющие такую трудоёмкость, известны.

Задача 15. (2×0.01) 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.

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

[14] Контрольные вопросы. Каким образом? Обсуждалась ли такая задача ранее?

[15] Если вы правильно понимаете обозначения, которые уже неоднократно использовались.

[16] *Выпуклой оболочкой* произвольного множества V точек векторного пространства называется наименьшее содержащее V подмножество, которое вместе с любой парой своих точек содержит и отрезок, их соединяющий.

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

Граф $G(V,E)$ называется *планарным*, если его можно вложить в плоскость, т. е. существует отображение $f: G \to \mathbb{R}^2$, 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.

Известная теорема Куратовского утверждает, что граф является **непланарным**, т. е. его нельзя вложить в плоскость, тогда и только тогда, когда в нем можно найти гомеоморфный образ[18] либо графа K_5 (полного пятивершинника), либо графа $K_{3,3}$ (три дома, три колодца).

Задача 16. (0.01) Используя теорему Куратовского, докажите, что язык кодировок непланарных графов принадлежит $\mathcal{NP}$.

Задача Д–7. (0.05) В некоторых источниках указывается, что непосредственно из теоремы Куратовского вытекает существование полиномиального алгоритма проверки непланарности. Если вы согласны с этим утверждением, то опишите соответствующий полиномиальный алгоритм.

Замечание. Высокий балл за это упражнение выставлен неслучайно. На самом деле, утверждение справедливо, но детали непосредственного использования критерия Куратовского не были известны вплоть до 1990 г. И это при том, с начала 70-х годов был известен **линейный алгоритм** проверки планарности. Но он основан на другом критерии планарности.

Аналогично, 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. Непланарные графы K_5 и $K_{3,3}$ можно вложить в тор, поскольку на плоскости удаётся 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 $K_{3,4}$ (полный 6-вершинник) или, тем более, с графом $K_{3,6}$ (три дома, шесть колодцев).

Задача Д–8. (0.01) Докажите или опровергните, что граф $K_{3,6}$ является торическим.

Тор удобно изображать прямоугольником с отождествлёнными противоположными сторонами.

Задача Д–9. (0.02) Покажите, что класс торических графов, так же как и класс планарных графов, принадлежит $\mathcal{NP}$.

NP-полные языки

Назовём предикат $L \in \mathcal{NP}$ *NP-полным*, если любой предикат из $\mathcal{NP}$ к нему полиномиально сводится. Такие предикаты естественно было бы считать "самыми сложными" в классе $\mathcal{NP}$, т. к. 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 $c(L)$.

Но существуют ли NP-полные предикаты вообще? С одной стороны, этот вопрос кажется тривиальным, ибо почти **по определению** имеет место

Предложение. *Язык TMSAT=*$\{(\langle M\rangle, x, 1^n[n_{unary}], 1^t[t_{unary}]) \mid \exists y \in \{0,1\}^n$, *такой что MT с описанием* $\langle M\rangle$ *принимает слово* $[x,y]$ *за* t *тактов*$\}$ *является NP-полным в $\mathcal{NP}$.*

Доказательство. Если $L \in \mathcal{NP}$, то для любого $x \in L$ существует "сертификат" $y \in \{0,1\}^{p(|x|)}$ и полиномиальная МТ R, такая что $R(x,y) = 1$. Тогда сводимость строится так: $x \to [\langle R\rangle, x, 1^{p(|x|)}, 1^{q(|m|)}]$, здесь $m = |x| + p(|x|)$ и $q(\cdot)$ — (полином) времени работы МТ R.

Приведённый пример NP-полного языка является не очень содержательным, поскольку определялся через те же МТ. В 1971 году С. Кук и Л. Левин предложили примеры комбинаторных NP-полных языков, с виду совершенно не связанных с моделями вычислений.

Самый известный NP-полный язык — это, конечно, ВЫПОЛНИМОСТЬ, который состоит из кодировок всех выполнимых булевых формул. Иначе говоря, для каждой формулы [19] 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 трёх 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.

Приведём 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 предположить, что мощности всех* A_i *равны* 2.

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

ВЕРШИННОЕ ПОКРЫТИЕ

ХРОМАТИЧЕСКОЕ ЧИСЛО. Даны неориентированный 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 неориентированный 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 $w: E \to$

[18] Т. е. граф полученный посредством подразбиения ребер исходного графа (добавления произвольного числа дополнительных вершин на ребра).

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

$\mathbb{Z}_+$ 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** $\psi(x_1, x_2, x_3) = (x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_1) \wedge (\neg x_2 \vee x_3)$ [зависящую от трех переменных и имеющую 3 дизъюнкта] и **невыполнимую КНФ** $\chi(x_1, x_2, x_3) = (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge \neg x_3$ [зависящую от трех переменных и имеющую 5 дизъюнктов].

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

Задача Д–10. (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 произвольную 3-КНФ ϕ, в которой в каждом дизъюнкте содержится не более трех литералов, причём 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 17. (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.**, а подсказкой являются две следующие задачи. Нужно специфицировать параметры, выделенные звёздочками, по кальке изложенной выше сводимости.

(*i*) Укажите для семейства $\mathcal{A}_\psi$ соответствующее $*?*$-**элементное** протыкающее множество.

(*ii*) Докажите, что мощность любого протыкающего множества для семейства $\mathcal{A}_\chi$ больше $*?*$.

Если использовать полный язык 3-ВЫПОЛНИМОСТЬ, то из построенной сводимости следует, что язык остаётся NP-полным, даже если все A_i имеют не более 3 элементов. Но оказывается, что язык остаётся NP-полным, даже если все A_i двухэлементные. Если отождествить эти пары элементов с рёбрами 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[20]. Во-первых, будем считать, что исходная КНФ дополнена до РОВНО-3-КНФ и в каждый её 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 18. (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.

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 19. (2×0.01) (*i*) Укажите для графа $\tilde{G}_\psi$ соответствующую $*?*$-клику.

(*ii*) Докажите, что мощность любой клики в графе $\tilde{G}_\chi$ меньше $*?*$.

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 20. (2×0.02) (*i*) Преобразуйте ψ в РОВНО-3-КНФ [в которой образовалось k 3-дизъюнктов] и вычис-

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

лите 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 21. (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 четвертую неделю: 25.02- 3.03 [0.07] Класс $\mathcal{PSPACE}$. Полные языки в $\mathcal{PSPACE}$ Литература: [К-Ш-В, АВ, GL]

Краткий конспект

Мы рассматривали предикаты, в которых используется только один тип квантора ($\exists$ для выделения $\mathcal{NP}$ и $\forall$ — для $co-\mathcal{NP}$, но нетрудно указать естественные языки в который нужно использовать несколько разных типов кванторов. Например, верно ли, что данная КНФ выполнима и является самой *короткой* среди всех КНФ, вычисляющих ту же булеву функцию? Возникает вопрос: что можно сказать о сложности таких предикатов?

Воспользуемся игровой интерпретацией сложностных классов[21]. Рассмотрим игру двух старых знакомых **А**ртура и **М**ерлина. Игрокам сообщается некоторая начальная позиция — слово x, и они делают ходы по очереди (w_1 — первый ход **A**, b_1 — первый ход **M** и т. д.). В нашей старой картинке считалось, что именно **M** предоставляет **A** слово сертификат, а потом **A** его отвергает или принимает. Сейчас оба игрока стали равноправными и характеризуются только порядком ходов. Каждый ход описывается словом длины $poly(|x|)$. Игра завершается после заранее заданного числа ходов. На самом деле, правило завершения игры может быть и сложнее. Если есть верхняя оценка на число ходов, то можно всегда дополнить число ходов до этой оценки, никак не учитывая дополнительные ходы при подведении результата игры. Результат игры описывается некоторым полиномиально разрешимым предикатом $W(x, w_1, b_1, \dots) \in \mathcal{P}$, истинность которого означает, что выиграл **A** (ничьих не бывает, так что ложность W означает, что выиграл **M**). Такие предикаты назовём **полиномиально разрешимыми**. Предикат W зависит от исходного слова и ходов, сделанных игроками: $w_1, w_2, \dots$ — **A** и, $b_1, b_2, \dots$ — **M**. Поскольку ¶ замкнут относительно дополнений, предикат $B(\cdot) = \neg W(\cdot)$, утверждающий выигрыш принадлежит $\mathcal{P}$.

Отсутствие ничьих и конечность числа ходов гарантируют при заданном x существование выигрышной стратегии либо для белых, либо для чёрных.[22]

Поэтому каждой игре отвечают два взаимно дополнительных множества:
$L_A = \{x|$ **A имеет выигрышную стратегию** $\}$ и
$L_M = \{x|$ **M имеет выигрышную стратегию** $\}$.

Рассмотренные сложностные классы можно определить как множества L_A (или L_M), соответствующие тем или иным видам игр. Например, получаем следующие классы.

Множества L_A (как, впрочем, и L_M) в играх *с отсутствием ходов* составляют класс $\mathcal{P}$ для полиномиально разрешимых игр.

Множества L_A для игр, в которых **A** делает 1-й ход и выигрывает, т. е. языки вида $\{x|\exists\, w_1\, W(x, w_1)\}$, составляют класс $\mathcal{NP}$ для полиномиально разрешимых игр.

Множества L_M для игр, в которых **M** делает 1-й ход и выигрывает, т. е. языки вида $\{x|\forall\, w_1\, B(x, w_1)\}$, являющиеся дополнениями NP-языков, составляют класс $co-\mathcal{NP}$ для полиномиально разрешимых игр.

Множества L_A для игр из двух ходов: первым ходит **A**, а затем ходит **M** и **A** выигрывает, т. е. языки вида $\{x|\exists\, w_1 \forall\, b_1\, W(x, w_1, b_1)\}$ образуют класс Σ_2^p в т. н. полиномиальной иерархии (ПИ) для полиномиально разрешимых игр. Словами: есть такой ход **A**, что, как бы ни сыграл **M**, **A** выигрывает.

Σ_k^p: множества L_A для игр из k ходов (в зависимости от чётности k последним ходит либо **M**, либо **A**) и **A** выигрывает. Другими словами, это множества вида

$$\{x : \exists\, w_1 \forall\, b_1\ \dots\ \mathsf{Q}_k\, y_k\, W(x, w_1, b_1, \dots)\}$$

(если k чётное, то $\mathsf{Q}_k = \forall$, $y_k = b_{k/2}$, если k нечётное, то $\mathsf{Q}_k = \exists$, $y_k = w_{(k+1)/2}$).

Π_k^p: множества L_M для игр из k ходов (в зависимости от чётности k последними ходят либо **M**, либо **A**) и **M** выигрывает. Другими словами, это множества вида

$$\{x : \forall\, w_1 \exists\, b_1\ \dots\ \mathsf{Q}_k\, y_k\, B(x, w_1, b_1, \dots)\}$$

(если k чётное, то $\mathsf{Q}_k = \exists$, $y_k = b_{k/2}$, если k нечётное, то $\mathsf{Q}_k = \forall$, $y_k = w_{(k+1)/2}$).

Все эти множества образуют т. н. *полиномиальную иерархию (ПИ)* и для неё можно доказать несколько утверждений о вложении классов Σ_k^p и Π_k^p (некоторые тривиальные, а некоторые условные, т. е. зависят, например, от совпадений классов на каких-то этажах), например, известное нам включение $\mathcal{P} \subseteq \mathcal{NP} \cap co-\mathcal{NP}$ (т. е. в новых обозначениях $\Sigma_0^p \subseteq \Sigma_1^p \cap \Pi_1^p$) переходит в Σ_k^p (или Π_{k+1}^p) $\subseteq \Sigma_{k+1}^p \cap \Pi_{k+1}^p$.

Мы не знаем строги ли эти включения. В принципе ПИ может, как говорят, *коллапсировать*, начиная с какого этажа все классы будут совпадать (например, возможно, что $\mathcal{P} \subset \mathcal{NP} = co-\mathcal{NP}$, и тогда вся ПИ коллапсирует на первый этаж).

Кроме того, вся ПИ принадлежит $\mathcal{PSPACE}$, т. е. классу языков, вычислимых на ленте полиномиального размера, к изучению которого мы переходим.

По определению, МТ M работает на памяти $S_M(n)$, если наиболее удалённая от начала ленты просматриваемая ячейка на всех входах длины n имеет номер $\leq S_M(n)$. Язык принадлежит классу $\mathcal{DSPACE}(s(n))$ (здесь $s(\cdot) : \mathbb{N} \to \mathbb{N}$), если он распознаётся МТ, работающей на памяти $s(n)$.

Формально $\mathcal{PSPACE} = \cup_{k\geq 0}\mathcal{DSPACE}(O(n^k))$. Аналогично определяется класс недетерминированных вычислений на ленте полиномиального размера: $N-\mathcal{PSPACE} = \cup_{k\geq 0}\mathcal{NSPACE}(O(n^k))$ (чуть ниже мы увидим, что $\mathcal{PSPACE} = N-\mathcal{PSPACE}$). Кроме того, при очень слабых условиях на функцию $s(n)$ верны (почти очевидные) включения:

$$DTIME(s(n)) \subseteq SPACE(s(n)) \subseteq NSPACE(s(n)) \subseteq DTIME(2^{s(n)})$$

Класс $\mathcal{PSPACE}$ — совсем не экзотика и, более того, неоднократно, но неявно возникал в ТРЯП. Например, проверка непустоты пересечения ДКА в двоичном алфавите принадлежит $\mathcal{PSPACE}$, хотя (предположительно) не входит в $\mathcal{NP}$.

Интуитивно $\mathcal{PSPACE}$ кажется значительно более сложным классом, чем $\mathcal{NP}$. Однако, в настоящее время не известно совпадает ли $\mathcal{P}$ и $\mathcal{PSPACE}$. Опять можно воспользоваться идеей полиномиальной сводимости, ведь в $\mathcal{PSPACE}$ есть полные языки относительно полиномиальной сводимости, поскольку рассуждать можно так же, как и для класса $\mathcal{NP}$: *Язык SPACE-TMSAT=$\{(\langle M\rangle, x, 1^t[t_{unary}]$ | детерминированная МТ с описанием $\langle M\rangle$ принимает слово x памяти $t\}$, по определению, является полным относительно полиномиальной сводимости в $\mathcal{PSPACE}$.* И вопрос о сложности $\mathcal{PSPACE}$ сводится к полиномиальной разрешимости любого полного языка.

Но, как и для класса $\mathcal{NP}$, мы бы хотели построить какой-то содержательный комбинаторный язык, полный относительно полиномиальной сводимости в $\mathcal{PSPACE}$. Оказывается, что в курсе ТРЯП мы уже сталкивались с такими сложными языками. Например, таким будет язык всех НКА (или, если угодно, регулярных выражения), которые не принимают некоторые слова, т. е. не принимают язык всех слов, а также указанный выше язык пересечение ДКА.

Для доказательства нам понадобится следующий общий неожиданный факт.

Теорема 2 (Savitch (Савич)) *[С точностью до некоторых технических деталей] для любой функции $f(\cdot) : \mathbb{N} \to \mathbb{N}$ и $f(n) \geq$*

[21] Здесь мы следуем изложению в **[К-Ш-В], §4**

[22] Формальное доказательство получается индукцией по числу ходов.

$\log n$ *выполнено* $\mathcal{NSPACE}(f(n)) \subseteq \mathcal{DSPACE}(f(n)^2)$, *т. е. любое недетерминированное вычисление на ограниченной ленте можно промоделировать детерминированных с не более, чем квадратичным увеличением памяти.*

Неформально доказательство использует т. н. *динамическое программирование.*

Вспомним, что конфигурацией МТ называется тройка [текущее состояние МТ; номер обозреваемой ячейки ленты; (бесконечное) слово (имеющее только конечное число непустых символов), записанное на ленте]. Заметим, что всего конфигураций на $f(n)$ ячейках может быть порядка $K = |Q|f(n)(const)^{f(n)}$. Отсюда вытекает, что любое вычисление нужно продолжать не более K тактов (оно либо остановится, либо зациклится). Таким образом, вопрос о том, существует ли принимающее вычисление для записанного на ленте слова, можно представить как **задачу нахождения пути на графе конфигураций с K вершинами из начальной конфигурации в какую-то принимающую.** Но весь вопрос в том, как проверить существование пути *экспоненциального* размера на ленте *полиномиальной длины.*

Здесь используется следующее соображение, которое мы встретим во многих, если не почти во всех, алгоритмах поиска путей в графе: (ориентированный) путь из $\leq 2k$ рёбер между вершинами s и t существует тогда и только тогда, когда есть "промежуточная" вершина v, такая что есть путь от s до v из $\leq k$ рёбер и путь от v до t из $\leq k$ рёбер. Итерируя эту конструкцию, можно, проводя вычисления на ленте полиномиальной длины, удостовериться, что экспоненциальное длинное принимающее вычисление для слова существует.

Обозначим граф конфигураций G. Итак (и это нужно уметь объяснять), для доказательства теоремы Савича достаточно показать, как на полиномиальной по входу ленте проверить, есть ли в графе G (экспоненциального по входу размера) путь между заданной парой вершин. Пусть $R(x, y, k)$ — предикат существования пути от x до y длины $\leq 2^k$. При этом обозначении принцип динамического программирования для кратчайших путей переписывается так $R(x, y, k)$ выполнен тогда и только тогда, когда для некоторой вершины z выполнено $R(x, z, k-1) \wedge R(z, y, k-1)$. Получаем следующий рекурсивный алгоритм: перебираем все z, для каждого вычисляем $R(x, z, k-1)$ запоминаем результат и на той же памяти вычисляем $R(z, y, k-1)$ и возвращаем конъюнкцию двух результатов. Базой рекурсии будет случай (тривиальный) случай $k = 0$.

На каждом шаге рекурсии требуется запомнить только текущую вершину z, что требует памяти $O(f(n))$ (именно здесь используется нижнее ограничение $f(n) \geq \log n$ — *почему? — число вершин в G порядка $f(n)2^{f(n)}$*). Глубина рекурсии составит также $O(f(n))$ (почему?), поэтому общая использованная память будет равна $O(f(n)^2)$.

В $\mathcal{PSPACE}$ роль "канонического полного языка" (типа SAT) играет язык $\mathcal{TQBF}$ ("totally quantified boolean formulae"), т. е. множество таких булевых формул $\varphi(x_1, \ldots, x_n)$, что для некоторого $x_1 \in \{0, 1\}$ найдётся $x_2 \in \{0, 1\}$, такое что для некоторого $x_3 \in \{0, 1\}$ найдётся ... (цепочка чередующихся кванторов по всем переменным), что $\varphi(x_1, \ldots, x_n)$ истинна.

Формально $\varphi(\cdot)$есть формула вида

$$\mathsf{Q}_1\, y_1 \ldots \mathsf{Q}_n\, y_n F(y_1, \ldots, y_n),$$

где $y_i \in \{0, 1\}$, F — некоторая логическая формула, а Q_i — либо $\forall$, либо $\exists$. По определению, $(\forall y_1\, W(y_1)) = (W(0) \wedge W(1))$, а $(\exists y_1\, W(y_1)) = (W(0) \vee W(1))$.

Каноничность состоит в том, что для любого фиксированного k язык $k - \mathcal{TQBF}$ т. е. выполнимость булевой формулы с k альтернациями (переменами) кванторов является полным на k-м этаже полиномиальной иерархии.

Сначала докажем утверждение (см. **[К-Ш-В], §4.2**).

Теорема 3 *$L \in \mathcal{PSPACE}$ тогда и только тогда, когда существует такая игра с полиномиальным от длины входного слова числом ходов и полиномиально вычислимым результатом, что $L = \{x \mid$ **A** имеет выигрышную стратегию$\}$.*

$\Longleftarrow$ Покажем, что язык, определяемый игрой, принадлежит $\mathcal{PSPACE}$. Пусть число ходов ограничено $p(|x|)$. Определим по индукции набор машин Тьюринга M_k для $k = 0, \ldots, p(|x|)$. Каждая M_k по заданному началу игры $x, w_1, b_1, \ldots$ длины k определяет наличие выигрышной стратегии у **A**. Последней в этом ряду машине $M_{p(|x|)}$ нужно просто вычислить предикат $W(x, w_1, \ldots)$. Машина M_k перебирает все возможные варианты $(k+1)$-го хода и консультируется с M_{k+1} по поводу окончательных результатов игры. Её оценка игры составляется очень просто: если текущий ход у **A**, то достаточно найти один ход, при котором M_{k+1} гарантирует выигрышную стратегию для **A**. Если текущий ход у **M**, то при всех вариантах M_{k+1} должна обнаружить выигрышную стратегию для **A**. Машина M_0 определяет наличие выигрышной стратегии для **A** в самом начале игры и для её работы нам нужно использовать всю последовательность машин M_k. Но каждая из этих машин использует небольшую (полиномиально ограниченную) память, так что весь процесс потребует лишь полиномиально ограниченной памяти.
Обратная импликации аналогична доказательству теоремы Савича.

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 22. (0.03) Покажите, что задача проверки неэквивалентности регулярных языков, заданных НКА, в унарном (однобуквенном) алфавите принадлежит классу $\mathcal{NP}$.

Задача 23. (0.03) Покажите, что задача проверки неэквивалентности регулярных языков, заданных НКА, в бинарном алфавите является полиномиально полной в классе $\mathcal{PSPACE}$.

Задание на пятую неделю: 4.03-10.03. [0.1] Вероятность и вероятностные алгоритмы Литература: [Кормен 1, Глава 36] [Кормен 3, Глав 34], [ДПВ. Глава 8]

Большинство приведённых ниже определений справедливы только для случая конечного множества элементарных исходов, хотя их можно обобщить на случай бесконечных вероятностных пространств, но это требует более серьёзной техники. В любом случае, следует быть особенно осторожным с определением случайных величин.

Вероятностное пространство. События.

1. **Вероятностным пространством** называется[23] множество Ω, элементы которого называются возможными или элементарными исходами $\omega \in \Omega$. На вероятностном пространстве задана функция $\mathbb{P}(\cdot) : \Omega \to [0, 1]$, называемая **вероятностным распределением**, для которой выполнено равенство $\sum_{\omega \in \Omega} \mathbb{P}(\omega) = 1$. Число $\mathbb{P}(\omega)$ понимается как вероятность исхода ω. **Событием** называется произвольное подмножество Ω. Соответственно, вероятность события $A \subseteq \Omega$ определяется формулой $\mathbb{P}(A) \stackrel{def}{=} sum_{\omega \in A} \mathbb{P}(\omega)$.

В качестве простейшего примера можно рассмотреть классическую, или наивную, теорию вероятности, когда все исходы полагаются равновероятными: например, при бросании симметричной монетки или игральной кости.

2. Если $\mathbb{P}(A) = 0$, то событие A называют **невозможным**. Если $\mathbb{P}(A) = 1$, то событие A называют **достоверным**. Если $\mathbb{P}(A \cap B) = 0$, то события A и B называются **несовместными**.

3. События A и B называют **независимыми**, если $\mathbb{P}(A \cap B) = \mathbb{P}(A) \cdot \mathbb{P}(B)$.

4. События $A_1, \ldots, A_n$ называются **попарно независимыми**, если любые два из них являются независимыми.

5. События $A_1, \ldots, A_n$ называются **независимыми в совокупности**, если для любого набора из этих событий $A_{i_1}, \ldots, A_{i_k}$ выполняется $\mathbb{P}(A_{i_1} \cap A_{i_2} \cap \cdots \cap A_{i_k} = \mathbb{P}(A_{i_1})\dot{\mathbb{P}}(A_{i_2}) \cdot \cdots \cdot \mathbb{P}(A_{i_k})$.

[23] Везде далее множество считается конечным.

6. **Условная вероятность** $\mathbb{P}(A \mid B)$ события A при условии B определяется из формулы $\mathbb{P}(A \cap B) = \mathbb{P}(A \mid B) \cdot \mathbb{P}(B)$. В частности, если $\mathbb{P}(B) > 0$, то справедливо равенство

$$\mathbb{P}(A \mid B) = \frac{\mathbb{P}(A \cap B}{\mathbb{P}(B)}.$$

7. Если $\mathbb{P}(A) > 0$ и $\mathbb{P}(B) > 0$, то выполнена **формула Байеса**

$$\mathbb{P}(A \mid B) = \mathbb{P}(A) \cdot \frac{\mathbb{P}(B \mid A}{\mathbb{P}(B)}.$$

.

8. Если события $B_1, \dots, B_n$ образуют **разбиение** вероятностного пространства Ω, т. е. $\Omega = B_1 \sqcup \dots \sqcup B_n$ и пересечения попарно пусты: $B_i \cap B_j = \emptyset$, $i \neq j$, то для любого события A выполнена формула полной вероятности:

$$\mathbb{P}(A) = \sum_{i=1}^{n} \mathbb{P}(A \mid B_i) \cdot \mathbb{P}(B_i).$$

Случайные величины

1. **Случайная величина** $\xi : \Omega \to \mathbb{R}$ — это числовая функция на вероятностном пространстве. Иначе говоря, случайная величина — это обычная числовая функция, на аргументах которой задано вероятностное распределение, так что можно говорить говорить о вероятности события, при котором она принимает конкретное значение.

2. **Математическим ожиданием** случайной величины ξ называется ее средневзвешенное значение

$$\mathbb{E}\xi \stackrel{def}{=} \sum_{\omega \in \Omega} \xi(\omega) \cdot \mathbb{P}(\omega).$$

3. По определению, математическое ожидание линейной комбинации случайных величин равно соответствующей линейной комбинации их матожиданий, т. е. $\mathbb{E}(a\xi + b\psi) = a\mathbb{E}(\xi) + b\mathbb{E}(\psi)$.

4. Следующий результат, тривиально вытекающий из определения математического ожидания, имеет обширные алгоритмические, комбинаторные и пр. приложения. Можно даже сказать, что это исходный пункт т. н. **вероятностного метода**.

Пусть для случайной величины ξ выполнено $\mathbb{E}\xi = A$, тогда существует исход ω, что $\xi(\omega) \geq A$ (равно как существует исход, при котором выполнено противоположное неравенство).

5. Две случайные величины ξ и ψ независимы, если для любых чисел c и d события $\xi^{-1}(c) = \{\omega \in \Omega : \xi(\omega) = c\}$ и $\psi^{-1}(d)$ независимы.

6. Математическое ожидание произведения независимых случайных величин равно произведению их математический ожиданий, т. е. $\mathbb{E}(\xi \cdot \psi) = \mathbb{E}\xi \cdot \mathbb{E}\psi$ (что можно проверить непосредственным вычислением).

7. **Дисперсией** $\mathbb{D}(\cdot)$ случайной величины называется средний квадрат ее отклонения от математического ожидания, т. е. $\mathbb{D}\xi = \mathbb{E}(\xi - \mathbb{E}\xi)^2$.

7. Для оценки вероятности отклонения *неотрицательной* случайной величины ξ от математического ожидания можно использовать **неравенство Маркова**

$$\mathbb{P}(\xi \geq a) \leq \frac{\mathbb{E}\xi}{a}.$$

8. Из неравенства Маркова и определения дисперсии немедленно следует неравенство Чебышева

$$\mathbb{P}(\mid \xi - \mathbb{E}\xi \mid \geq a) \leq \frac{\mathbb{D}\xi}{a^2}.$$

9. Неравенство больших уклонений. Обозначим через ξ_n случайную величину, равную количеству выпавших орлов после n подбрасываний "честной" монеты, а через $x_n = \frac{\xi_n}{n}$ — частоту выпавших орлов. При больших n частота с очень большой вероятностью оказывается близкой к $\frac{1}{2}$. Имеет место следующая оценка, которую иногда называют **неравенством Чернова**. Она очень удобна в приложениях вероятностного метода в комбинаторике, а также в теории вычислений.

$$\mathbb{P}\left(\mid \xi_n - \frac{n}{2} \mid > \varepsilon n\right) = \mathbb{P}(\mid x_n - \frac{1}{2} \mid > \varepsilon) < 2\exp -2\varepsilon^2 n$$

Имеют место ещё несколько неравенств подобного рода, если монетка несимметричная. Пусть орел выпадает в вероятностью p. Тогда

$$\mathbb{P}(x_n < (1-\delta)p) < \exp\left(-\frac{\delta^2 p}{2}\right), \qquad 0 \leq \delta \leq 1,$$

$$\mathbb{P}(x_n > (1+\delta)p) < \exp\left(-\frac{\delta^2 p}{3}\right), \qquad 0 \leq \delta \leq 1,$$

$$\mathbb{P}(x_n > (1+\delta)p) < \exp\left(-\frac{\delta p}{3}\right), \qquad \delta > 1.$$

Приведём набросок короткого доказательства немного более сильного неравенства[24].

Теорема 4 *Пусть $X_1, \dots, X_n$ — последовательность дискретный независимых случайный величин с нулевым средним $\mathbb{E}X_i = 0$, $i = 1, \dots, n$, таких что $|X_i| \leq 1$, $i = 1, \dots, n$. Обозначим $X = \sum_{i=1}^n X_i$ и пусть σ — дисперсия X. Тогда для всех $0 \leq \lambda \leq 2\sigma$ выполнено неравенство $\mathbb{P}(|X| \geq \lambda\sigma) \leq 2\exp(-\lambda^2/4)$.*

Доказательство. Ввиду симметрии достаточно обосновать неравенство $\mathbb{P}(X \geq \lambda\sigma) \leq \exp(-\lambda^2/4)$. Для любого $t \in (0,1)$ выполнено: $\mathbb{P}(X \geq \lambda\sigma) = \mathbb{P}(tX \geq t\lambda\sigma) = \mathbb{P}[\exp(tX) \geq \exp(t\lambda\sigma)] \stackrel{\text{нер-во Маркова}}{\leq} \frac{\mathbb{E}[\exp(tX)]}{\exp(t\lambda\sigma)}$. Осталось оценить сверху числитель, т. е. $\mathbb{E}[\exp(tZ)]$ для $|Z| \leq 1$, $\mathbb{E}Z = 0$ и $0 \leq t \leq 1$. Имеем $\mathbb{E}[\exp(tZ)] \stackrel{def}{=} \sum_{j=1}^{\infty} p_j \exp(tz_j) = \sum_{j=1}^{\infty} p_j(1 + tz_j + \frac{1}{2!}(tz_j)^2 + \dots) = \underbrace{\sum_{j=1}^{\infty} p_j}_{A} + \underbrace{\sum_{j=1}^{\infty} p_j z_j}_{B} + \underbrace{\sum_{j=1}^{\infty} p_j(\frac{1}{2!}(tz_j)^2 + \frac{1}{3!}(tz_j)^3 + \dots}_{C}$. Но $A = 1$ (сумма всех вероятностей), $B = \mathbb{E}Z = 0$, а C (поскольку $|tz_j| \leq 1$) не больше, чем $\sum_{j=1}^{\infty} p_j (tz_j)^2 (\frac{1}{2!} + \frac{1}{3!} + \dots) \leq t^2 \sum_{j=1}^{\infty} p_j (tz_j)^2 = t^2 \mathbb{D}Z$. Или все вместе: $\mathbb{E}[\exp(tZ)] \leq 1 + t^2 \mathbb{D}(Z)$.

Возвращаемся к доказательству неравенства Чернова: $\mathbb{E}[\exp(tX)] = \mathbb{E}[\exp(t(X_1 + \dots + X_n))] = \mathbb{E}[\prod_{i=1}^n \exp(tX_i)] = \prod_{i=1}^n \mathbb{E}[\exp(tX_i)] \leq \prod_{i=1}^n [1 + t^2 \mathbb{D}(X_i)] \stackrel{\text{исп. нер-во } 1+x \leq \exp(x),\, x \geq 0}{\leq} \prod_{i=1}^n \exp[t^2 \mathbb{D}(X_i)] = \exp(t^2\sigma^2)$

Контрольный вопрос. В последней цепочке неравенств два раза использовалась независимость X_i. Укажите эти места.

Таким образом, получено неравенство $\mathbb{P}(X \geq \lambda\sigma) \leq \frac{\exp(t^2\sigma^2)}{\exp(t\lambda\sigma)} = \exp[t\sigma(t\sigma - \lambda)]$, что при выборе $t = \frac{\lambda}{2\sigma}$ даёт искомое неравенство Чернова.

Важное замечание. **Число задач по этой теме существенно увеличено. Большинство из них очень простые, и мы постараемся разобрать существенную их часть на семинарах и на дополнительном курсе. Дело в том, что вероятностные аспекты становятся все более существенными не только в теоретической, но и в практической "алгоритмистике", так что полезно буквально с первых шагов привыкнуть к этому, а при разработке и анализе алгоритмов к случайности следует относиться как к потенциально важному ресурсу. Но осознать, в чем, собственно, этот ресурс заключается, можно только решая задачи.**

Задача 24. (4×0.01) Подбрасываем "честную" монету 10 раз. Подсчитайте вероятности следующих событий:

(i) число выпавших "орлов" равно числу "решек";

(ii) выпало больше "орлов" чем "решек";

(iii) при $i = 1, \dots, 5$ одинаковы результаты i-го и $11-i$-го бросаний;

(iv) "орел" выпал не менее четырех раз подряд.

Задача 25. (0.01) Вычислите условную вероятность, что при бросаний двух игральных костей на первой выпало шесть, а сумма равна семи.

[24] Автор Van Vu.

Задача 26. (0.01) При двух бросках игральной кости выпало X_1 и X_2, соответственно. Вычислите $\mathbb{E}[\max\{X_1, X_2\}] + \mathbb{E}[\min\{X_1, X_2\}]$ (чем проще, тем лучше).

Задача 27. (2×0.03)

(i) Найдите математическое ожидание числа бросаний кости до первого выпадения двух шестерок.

(ii) Симметричную монетку бросают неограниченное число раз. Какая из последовательностей встретится раньше с большей вероятностью: POP или PPO?

Задача 28. (2×0.02)

(i) Из бара на улицу выходит захмелевший турист. В одном конце улицы находится его гостиница, в другом — полицейский участок. Вдоль улицы горят фонари и турист идёт от одного фонаря к другому случайным образом: с вероятностью p в сторону гостиницы, с вероятностью $1 - p$ в сторону участка. Найти вероятность того, что в конце концов он дойдет до гостиницы. Считайте, что фонари пронумерованы от 0 (гостиница) до n (участок), и бар находится у фонаря с номером m. Вычислите значение вероятности для $n = 10$, $m = 5$, $p = 3$.

(ii) При тех же условиях найдите среднее расстояние[25], которое пройдёт турист.

Задача 29. (0.01) В двух спичечных коробках имеется по n спичек. Каждый раз, когда человеку нужна спичка, он берет её из случайного коробка (равновероятно). Найдите вероятность того, что когда один коробок опустеет, а в другом останется ровно k спичек.

Задача 30. (0.01) Независимы ли события: при броске кубика "выпало чётное число" или "выпало число очков, кратное трем"?

Задача 31. (0.01) Покажите, что из попарной независимости случайных величин не следует их независимость в совокупности. Приведите явный контрпример.

Задача 32. (0.01) Найти математическое ожидание числа простых циклов длины r в случайном графе на n вершинах. Считаем, что в графе ребра между каждой парой вершин независимо генерируются с вероятностью p. Такая модель случайного графа называется **моделью Эрдёша-Реньи**.

Задача 33. (0.02) Найдите математическое ожидание и дисперсию числа простых циклов длины r в случайной перестановке n элементов, в предположении, что все возможные перестановки равновероятны.

Задача 34. (2×0.02)

(i) Имеется генератор случайных битов, который выдает 0 или 1 с вероятностью $\frac{1}{2}$. Предложите алгоритм, который, используя данный генератор, возвращает 0 с вероятностью $\frac{1}{3}$ и 1 с вероятностью $\frac{2}{3}$. Оцените время работы вашего алгоритма в среднем и в худшем случае.

(ii) То же самое, но наоборот: из $\frac{1}{3}$ получить $\frac{2}{3}$.

[25] Измерять расстояние можно в фонарях по аналогии в измерении в попугаях в известном мультфильме.

Задача 35. (0.02) Предложите вероятностный алгоритм, который по заданному графу G генерирует случайное остовное дерево данного графа (с равномерным распределением).

Задача 36. (0.02).

Два будущих нобелиата (дальше идет прямая цитата): "...придумали эксперимент, логика и красота которого восхитили бы Грегора Менделя. В общих чертах он заключается в следующем.

Вырастим колонию в миллиард бактерий, начав с одной-единственной клетки, например обожаемой бактериологами кишечной палочки E. coli. (E. coli заселяет ваш кишечник в течение нескольких часов после вашего рождения, переключается из аэробного режима в анаэробный, прилипает к слизистой оболочке и живёт там до самой нашей смерти.) Предположим, что при воздействии на такую колонию некоторого смертельного для клеток фактора X (например, бактериофага T1 — лучшего друга E. coli) существует всё же маленькая вероятность, скажем, $p_1 = 0.02$, что какая-то клетка выживет. Выживание клетки есть результат мутации, придающей ей некое новое свойство П, проявляющееся в устойчивости мутанта к фактору X. Более того, и это существенно, свойство П передаётся всем потомкам нашей П-клетки.

Если применить эту процедуру, скажем, к тысяче различных колоний, то примерно в двадцати из них обнаружатся уцелевшие клетки. Забудем про вымершие колонии и посмотрим, в скольких из оставшихся уцелели две (или более) бактерии. Существуют две гипотетические альтернативы..."

Вычислите (приведите оценки) $p_2^{Lam}, n^{Lam}, p_2^{rand}, n^{rand}$ и обоснуйте их. **Целью эксперимента была дискриминация между двумя возможными моделями эволюции.**

(i) *Ламарковская*[26] *адаптация.* Если свойство П развивается как ответ на фактор X *после* его введения, то вероятность обнаружить две уцелевшие клетки равна $p_2^{Lam} =$??? (адаптивное) [напишите формулу или число и обоснуйте].

В этом случае *наличие* $n^{Lam} =$??? *колоний с двумя уцелевшими клетками крайне маловероятно.*

(ii) *Чистая случайность.* Если некоторые бактерии мутируют *до* введения фактора X, то $p_2^{rand} =$???, так что, скорее всего мы увидим примерно $n^{rand} =$??? колоний *с двумя уцелевшими клетками.*

Резюме. Если n^{Lam} и n^{rand} достаточно различаются, то это будет состоятельным аргументом в пользу той или иной концепции мутаций (и эволюции) со всем вытекающими последствиями.

Задача 37. (0.02) Используя оценку Чернова, оцените вероятность события, что при случайном и независимом распределении $3n \ln n$ шаров по n урнам максимальное число шаров в каждой урне не превосходит $2 \ln n$.

6. Темы 6-й недели 11.03-17.03

Вероятностные алгоритмы. Классы $\mathcal{RP}$, $\mathcal{BPP}$, $\mathcal{ZPP}$. Проверка полиномиальных тождеств Поиск паросочетаний в графах Лемма Шварца–Зиппеля. Дерандомизация Литература: [Кормен 2, §5 и дополнение C] [Кормен 1, §6], [GL], [ДПВ], [К-Ф], [К-Ш-В]

[26] В честь автора концепции Жана Батиста Ламарка (1744–1829).

Краткий конспект

В этом задании мы рассмотрим процедуры, использующие *рандомизацию*. Но мы не будем делать *никаких априорных предположений о входном распределении*,

Мы уже обсуждали как минимум три подобные процедуры (быструю сортировку, поиск медианы и проверку простоты), использующие бросание монетки (вероятностный алгоритм) и перейдём теперь к их формальному описанию. К сожалению, из-за недостатка времени мы успеем только ознакомиться с определениями и решить несколько задач. Но можно без преувеличения сказать, что вероятностные подходы к алгоритмам являются стержнем многих современных исследований[27]. Представить без них мир алгоритмов совершенно невозможно, так что всем заинтересованным лицам будет нелишне продолжить изучение этого подхода самостоятельно.

И основной лозунг, под которым оформлено это задание звучит так.

К возможности использования случайных битов нужно относиться как к дополнительному вычислительному ресурсу, позволяющему иногда существенно понизить трудоёмкость и концептуально упростить процедуру.

Основным источником таких возможностей является (гипотетическая) возможность алгоритмического порождения "случайных объектов" в конкретных "универсумах", так сказать, "псевдослучайные генераторы". Мы будем использовать простейшие варианты: "орлянку", выбор случайного натурального числа на отрезке $[1, N]$, выбор случайного ребра в графе, выбор случайной плоскости, проходящей через начало координат и т.д. Практически это рутинные программистские операции, используемые часто рефлекторно. Хотя даже на практическом уровне серьёзно обсуждаются вопросы о качестве этих генераторов.

А на теоретическом уровне даже предположение о существовании некоторых таких объектов типа доказуемых случайных генераторов оказывается эквивалентным гипотезам о сложности вычислений.

В соответствие с идеологией нашего курса, в идеале, в каждом случае, когда нам нужно породить "случайный объект", мы должны указывать конкретную процедуру порождения, уметь проверять (доказывать), что она действительно порождает нужные объекты, и уметь оценивать её трудоёмкость. Эти вопросы достаточно тонкие и, более того, многие из них пока не имеют удовлетворительных ответов.

Вероятностная Машина Тьюринга (ВМТ) представляет из себя обычную МТ, которой в некоторых состояния разрешено совершать переходы в зависимости от бросания монеты. В отличии от недетерминированной МТ легко представить себе практическую реализацию такой конструкции. Более того, как утверждают некоторые апологеты теории вероятностей, иных устройств в природе просто не существует. Будем считать, что используются стандартные монетки для игры в орлянку[28], так что каждое вычисление ВМТ на входе x полностью определено, если считать (по аналогии с определением класса $\mathcal{NP}$), что одновременно с x на вход детерминированной МТ подаётся (вообще говоря, бесконечное) $\{0,1\}$-слово.

Как определить, что слово или язык принимается ВМТ?

В соответствии с определением ВМТ любое вычисление имеет некоторую вероятность.

Будем говорить, что язык $L \subset \Sigma^*$ принимается ВМТ M в *СЛАБОМ смысле [по стандарту МОНТЕ-КАРЛО]*, если для любого слова $x \in \Sigma^*$ вероятность получения ошибочного ответа на вопрос: ($x \in L$?) не превосходит $\frac{1}{3}$. Иначе говоря, если $x \in L$, то M с вероятностью, не меньшей $\frac{2}{3}$ ПРИНИМАЕТ x; а если $x \notin L$, то M с вероятностью, не меньшей $\frac{2}{3}$ ОТВЕРГАЕТ x.

Будем говорить, что язык $L \subset \Sigma^*$ принимается ВМТ M в *СИЛЬНОМ смысле [по стандарту ЛАС-ВЕГАС]*, если она дает *с вероятностью* 1 правильный ответ для любого слова $x \in \Sigma^*$. Это, в частности, означает, что M не может за конечное число шагов принять какое-нибудь слово $x \notin L$.

[27] Достаточно посмотреть на изменения, внесённых во 2-е издание Кормена.

[28] Монетки у которых, скажем, вероятность выпадения герба является каким-нибудь *невычислимым* числом, в принципе можно использовать, для распознавания невычислимых языков.

Языки, принимаемые ВМТ в СЛАБОМ смысле за *полиномиальное в среднем число шагов*, образуют класс $\mathcal{BPP}$.

Языки, принимаемые ВМТ в СИЛЬНОМ смысле за *полиномиальное в среднем число шагов*, образуют класс $\mathcal{ZPP}$.

Наконец, языки, для которых удаётся построить ВМТ, которая за полиномиальное в среднем число шагов принимает каждое слово из языка с вероятностью $\geq \frac{1}{2}$ и отвергает любое слово, не входящее в язык, образуют промежуточный класс $\mathcal{RP}$.

По построению, классы $\mathcal{BPP}$ и $\mathcal{ZPP}$ замкнуты относительно дополнения и $\mathcal{BPP} \supseteq \mathcal{RP} \supseteq \mathcal{ZPP} \supseteq \mathcal{P}$. В основном, мы будем изучать $\mathcal{BPP}$. Рекомендуется посмотреть [К-Ф, гл. 4–5, §6.2] или книгу [К-Ш-В]. В последней, в разделах 1.3-1.4 дано определение класса $\mathcal{BPP}$, приведён вероятностный тест простоты Миллера-Рабина, показано, что $\mathcal{BPP} \subset \Sigma_1^p \cap \Pi_1^p$ (т. е. принадлежит второму этажу полиномиальной иерархии). В разделе 12.2 построена вероятностная полиномиальная сводимость вычисления дискретного логарифма к задаче разложения числа на множители (задаче факторизации). Кроме того, сами квантовые вычисления являются аналогом вероятностных вычислений со специально определенным правилом вычисления вероятности. При этом, однако, оказывается, что квантовые компьютеры позволяют решать, например, задачу факторизации за полиномиальное время. Как известно, никто пока не знает, есть ли полиномиальный (детерминированный или вероятностный) классический алгоритм для этой задачи. Кроме того, никто пока не умеет решать на квантовом компьютере какую-нибудь NP-полную задачу за полиномиальное время.

Задача 38. $(0.01 + 0.01)$ Покажите, что класс $\mathcal{BPP}$ не изменится, если
(i) константу стандарта Монте-Карло $\frac{1}{3}$ заменить на любое число, строго меньшее $\frac{1}{2}$, а
(ii) **полиномиальное в среднем число шагов** заменить на **полиномиальное число шагов**.

Последняя задача позволяет дать определение класса $\mathcal{BPP}$ по аналогии с классом $\mathcal{NP}$.

Предикат L принадлежит классу $\mathcal{BPP}$, если существуют такие полином $q(\cdot)$ и предикат $R(\cdot,\cdot) \in \mathcal{P}$, что

$L(x) = 1 \implies$ *доля слов r длины $q(|x|)$, для которых выполнено $R(x,r)$, больше $2/3$;*

$L(x) = 0 \implies$ *доля слов r длины $q(|x|)$, для которых выполнено $R(x,r)$, меньше $1/3$.*

Иначе говоря, на вход недетерминированной МТ подается слово-вход x и слово-подсказка r, и x принимается, если и только если при некоторой (не слишком длинной) подсказке принимается пара (x,r). Соответственно, ВМТ читает слово-вход x, а роль слова-подсказки r выполняют результаты бросания монетки в процессе вычисления, причем слово x принадлежит языку, если пары (x,r) принимаются для фиксированной доли C_1 подсказок (бросаний монеты) и отвергается, если число допустимых пар (x,r) меньше некоторой фиксированой доли C_2. По определению, константы C_1 и C_2 должны иметь "зазор" $C_1 - C_2 > \varepsilon$. Если последнее требование опустить (т. е. положить $C_1 = C_2 = \frac{1}{2}$), то вычислительные возможности ВМТ неизмеримо возрастают, а класс распознаваемых на таких ВМТ языков называется $\mathcal{PP}$. В частности, $\mathcal{NP} \subseteq \mathcal{PP}$.

Задача Д–11. (7×0.01) Обоснуйте включения (i) $\mathcal{P} \subseteq \mathcal{ZPP}$;
(ii) $\mathcal{ZPP} \subseteq \mathcal{RP} \cap co-\mathcal{RP}$;
(iii) $\mathcal{RP} \cup co-\mathcal{RP} \subseteq \mathcal{BPP}$;
(vi) $\mathcal{RP} \subseteq \mathcal{NP}$ и $co-\mathcal{RP} \subseteq co-\mathcal{NP}$;
(v) $\mathcal{BPP} \subseteq \mathcal{PP}$;
(vi) $\mathcal{NP} \cup co-\mathcal{NP} \subseteq \mathcal{PP}$;
(vii) $\mathcal{PP} \subseteq \mathcal{PSPACE}$.

Проверка (полиномиальных) тождеств является одним из наглядных и убедительных примеров нетривиального использования вероятностных алгоритмов. В основе подхода лежит лемма, смысл которой в том, что если полином не равен нулю тождественно, то он не может слишком часто обращаться в нуль, например, в точках целочисленной решётки[29].

Лемма 1 (Шварц- Зиппель) *Пусть $f(x_1,\dots,x_n)$, не равный тождественно нулю полином степени не выше k по каждой переменной*[30]*, и пусть принимающие целые значения случайные величины $\xi_1,\dots,\xi_n$ независимо и равномерно распределены на отрезке $[0,N-1]$.*

Тогда $\mathbb{P}\{f(\xi_1,\dots,\xi_n)=0\}\le\frac{kn}{N}$.

Это утверждение мы обсудим на лекции, и на эту тему будут задачи как в ближайших контрольных, так и в финальном тесте.

Доказательство леммы *проводится индукцией по числу переменных n.* Утверждение верно при $n=1$, поскольку нетривиальный полином степени k имеет не более k корней. При $n>1$ разложим f по переменной x_1: $f=f_0+f_1x_1+\ldots+f_tx_1^t$, где полиномы $f_0,\dots,f_t$ не зависят от x_1, а f_t не равен нулю тождественно. Тогда по формуле полной вероятности $Prob\{f=0\}=Prob\{f=0\mid f_t=0\}Prob\{f_t=0\}+Prob\{f=0\mid f_t\ne0\}Prob\{f_t\ne0\}\le Prob\{f_t=0\}+Prob\{f=0\mid f_t\ne0\}$. Первый член оценивается по индуктивному предположению, а второй — не больше, чем $\frac{k}{N}$, поскольку на каждом отрезке $[(0,\xi_2,\dots,\xi_n),(N-1,\xi_2,\dots,\xi_n)]$ полином $f=f_0+f_1x_1+\ldots+f_tx_1^t$ с ненулевым старшим коэффициентом f_t может иметь не более $t\le k$ корней, так что $Prob\{f=0\}\le\frac{k(n-1)}{N}+\frac{k}{N}$.

Задача 39. (4×0.01) Проверьте матричное равенство $C=AB$, где A,B,C– $n\times n$ матрицы, имеющие целочисленные элементы, не превышающие по абсолютной величине h, используя рандомизацию.

Пусть x— случайный n-мерный вектор, компоненты которого независимые целые числа, равномерно выбранные из интервала $[0,1,\dots,N-1]$. Проверка равенства состоит в вычислении $A(Bx)=Cx$: если это равенство справедливо, то вы предполагаете, что исходное равенство верное, иначе вы сигнализируете об ошибке. Заметим, что каждую такую проверку можно выполнить за $O(n^2)$ операций над $O(\log(nh^2))$-разрядными числами, а любой сигнал об ошибке говорит о том, что исходное равенство неверное. С другой стороны, если проверка прошла успешно, то возможно, что исходное равенство неверное, но мы неудачно подобрали тестовый вектор x.

(*i*) Каким нужно выбрать N, чтобы вероятность ошибки вашей процедуры была меньше заданной вероятности p?

(*ii*) Тот же вопрос, если разрешается проводить несколько независимых проверок, а минимизировать нужно общую *битовую сложность* вычислений.

(*iii*) Сравните битовую сложность вероятностных процедур с стандартным детерминированным алгоритмом перемножения матриц для $n=10000;h=2^{15};p=0.001$

(*iv*) Для дальнейшей экономии вы решили использовать проверку $(A(Bx)x)=(Cx)x$ или проверку $(A(Bx)y)=(Cx)y$, где n-вектор y выбирается независимо от x и имеет те же характеристики. Как изменится для этих случаев N?

[29]Это утверждение известно как Schwartz-Zippel Lemma. На самом деле, Шварцу, видимо, принадлежит вероятностная интерпретация, поскольку сам факт давно известен (см., например, главу "Сравнения" в книге Боревича и Шафаревича "Теория чисел"), но ему не придавали вероятностной интерпретации.

[30]Лемма остаётся верной, если считать, что k — это суммарная степень по совокупности переменных.

Язык 2-ВЫПОЛНИМОСТЬ состоит из выполнимых КНФ, в которых каждый дизъюнкт содержит не более двух литералов. Вы знаете, что задачу можно решать за линейное время, используя линейный алгоритм выделения сильно связных компонент в орграфах (об этом мы говорили на последней лекции). В этой задаче мы построим быстрый вероятностный алгоритм для проверки выполнимости 2-КНФ. Пусть 2-КНФ имеет n литералов и m дизъюнктов.

Алгоритм случайного поиска для языка 2-КНФ. Сначала всем переменным присваивается значение TRUE. На каждой итерации, пока формула невыполнима, берётся произвольный невыполненный дизъюнкт, в нем равновероятно выбирается произвольный литерал и его логическое значение обращается. Этот процесс напоминает случайное блуждание и в принципе может продолжаться бесконечно (например, если взять невыполнимую КНФ). Однако для выполнимой 2-КНФ можно получить полиномиальные оценки среднего числа итераций. Дело в том, что *число отличий* между текущим набором логических значений переменных и их значениями в некотором произвольном (но фиксированном) выполняющем наборе (существующем по предположению) изменяется на каждой итерации с вероятностью $\frac12$ на 1. Таким образом, наш алгоритм можно интерпретировать как случайное блуждание на отрезке [0,1,. . . ,n].

Задача Д–12. $(0.04+0.01)$ (*i*) Покажите, что для выполнимой 2-КНФ среднее число итераций алгоритма (математическое ожидание числа итераций) равно $O(n^2)$.

(*ii*) Пусть пункт (*i*) справедлив (а больше ничего о языке 2-КНФ неизвестно). В какой из вероятностных классов, определённых выше, попадает тогда язык 2-КНФ?

Задача о минимальном разрезе в неориентированном графе $G=(V,E)$ заключается в том, чтобы разбить вершины графа на два дизъюнктных подмножества $(S,\bar S)$, $S\ne V$, $S\ne\varnothing$ так, чтобы минимизировать число рёбер с концами в разных долях. Конечно, для ее решения можно применить потоковый алгоритм, но мы рассмотрим **вероятностный алгоритм Каргера**. При этом основной будет операция стягивания ребра (кратные ребра остаются, а петли удаляются). Граф, полученный стягиванием ребра $(x,y)\in E$, обозначим $G/(x,y)$. Первоначальная идея заключается в следующем: при стягивании рёбер величина минимального разреза не убывает (пока в графе остается не менее двух вершин), так что если стянуть все ребра $\{e_1,\dots,e_p\}$, не входящие в минимальный разрез, то останется пара вершин, соединённая k рёбрами, где k — величина минимального разреза в G. Остаётся понять, как часто реализуется подобная ситуация, если ребра стягиваются случайно.

Задача 40. $(2\times0.02+0.01)$ (*i*) Покажите, что вероятность того, что случайно выбранное ребро в графе входит в минимальный разрез не превышает $\frac{2}{|V|}$.

Из предыдущей задачи вытекает следующий вероятностный алгоритм определения минимального разреза:

MINCUT $[G(V,E),|V|=n]$

$G_0\leftarrow G$; $i\leftarrow0$;

while $|V(G_i)|>2$ **do**

В G_i выбираем случайное ребро e_i (с равномерным распределением на ребрах G_i).

$G_{i+1}\leftarrow G_/e_i$; $i\leftarrow i+1$;

end while

Комментарий. На выходе из цикла получаем (мульти)граф $\tilde G$, имеющий две вершины, соединённые рёбрами, иногда отвечающими разрезу в исходном графе.

return Разрез в исходном графе G, отвечающий разрезу в $\tilde G$.

Времы работы алгоритма $O(n^2)$.

(*ii*) Покажите что MINCUT выдает минимальный разрез с вероятностью $\ge\frac{2}{n(n-1)}$.

(*iii*) Покажите, что если независимо повторить процедуру MINCUT n^2 раз, то минимальный разрез будет найден с вероятностью > 0.85.

В следующей задаче мы покажем, что если привлечь дополнительные соображения, то можно понизить трудоёмкость до $O(n^2 \log^{O(1)} n)$. При этом алгоритм столь же прост и допускает параллелизацию.

Описанная выше процедура поиска минимального разреза последовательно выбирает случайные ребра и стягивает их концы до тех пор, пока в графе не останутся две вершины, соединённые (кратными) рёбрами. Если при выборе случайных рёбер мы ни разу не выбирали ребра разреза, то мы получаем ответ. Выше мы показали, что вероятность на i-м шаге выбрать ребро, входящее в разрез, равна $p_i = \frac{2}{n-i}$, отсюда вероятность того, что за i шагов не будет выбрано ни одно ребро, входящее в минимальный разрез (мы будем говорить, что выбранные *ребра не задевают разрез*), равна $P_i = (1-p_1)(1-p_2)\dots(1-p_i)$. Мы хотим ускорить алгоритм. Заметим, что чем больше рёбер мы стягиваем, тем больше вероятность, что следующее выбранное ребро заденет минимальный разрез. Поэтому новая идея, которую мы хотим исследовать, заключается в том, чтобы стягивать ребра до какого-то порога, пока вероятность попадания в разрез еще достаточно мала, а дальше использовать рекурсию. Из формулы для P_i видно, что если стянуть $n/2$ случайных ребер, то они с вероятностью $\geq \frac{1}{4}$ не заденут минимальный разрез.

Блок-схема нового алгоритма приведена ниже. Процедура использует подпрограмму СТЯГИВАНИЕ (G,k), которая стягивает ребра до тех пор пока число вершин не уменьшится ниже порога k (при стягивании произвольного ребра число вершин уменьшается на единицу).

СТЯГИВАНИЕ (G,k)
for $i := n$ **downto** k
В G выбираем случайное ребро e
(с равномерным распределением
на ребрах).
$G \leftarrow G/e$
endfor
return G

МИН-РАЗРЕЗ (G)
if в G больше восьми вершин **then**
Повторить 4 раза процедуру
$X_1 \leftarrow$ МИН-РАЗРЕЗ [СТЯГИВАНИЕ $(G, \frac{n}{2})$];
$X_2 \leftarrow$ МИН-РАЗРЕЗ [СТЯГИВАНИЕ $(G, \frac{n}{2})$];
$X_3 \leftarrow$ МИН-РАЗРЕЗ [СТЯГИВАНИЕ $(G, \frac{n}{2})$];
$X_4 \leftarrow$ МИН-РАЗРЕЗ [СТЯГИВАНИЕ $(G, \frac{n}{2})$];
return $\min\{X_1, X_2, X_3, X_4\}$
else находим минимальный разрез вручную.

Задача Д–13. $(0.01 + 0.01 + 0.04 + 0.01)$

(*i*) Запишите рекуррентную оценку сложности $T(n)$ вычисления функции МИН-РАЗРЕЗ (G) для графа с $|V| = n$ вершинами.

(*ii*) Найдите Θ-асимптотику $T(n)$ (для этого нужно сначала оценить трудоёмкость процедуры СТЯГИВАНИЕ (G,k)).

Оценим теперь с какой вероятностью $\mathbb{P}(n)$ алгоритм МИН-РАЗРЕЗ(G) выдает минимальный разрез для графа G с n вершинами. Вероятность успеха равна вероятности того, что хотя бы один рекурсивный вызов дал корректный ответ. Таким образом, получаем: $\mathbb{P}(n) = 1 - \left(1 - \frac{1}{4}\mathbb{P}(\frac{n}{2})\right)^4$, откуда следует, что $\mathbb{P}(n) \geq \mathbb{P}(\frac{n}{2}) - \frac{3}{8}\mathbb{P}(\frac{n}{2})^2$. Считая, что n является степенью двойки, обозначим $p_k = \mathbb{P}(2^k)$. По определению, p_k равно вероятности успеха, если потребовалось k рекурсивных вызовов процедуры МИН-РАЗРЕЗ. В частности, $p_0 = 1$.

Получаем рекурсию $p_{k+1} = 1-(1-\frac{1}{4}p_k)^4$. Откуда, если раскрыть скобки, следует: $p_{k+1} \geq p_k - \frac{3}{8}(\mathcal{P}_k)^2$.

(*iii*) Приведите как можно более точную оценку снизу рекурсии: $p_0 = 1; p_{k+1} = p_k - \frac{3}{8}(p_k)^2$ вида $p_k = \Omega(f(k))$.

Подсказка. Предположите, что $p_{k+1} - p_k \approx \frac{dp}{dk}$, и оцените порядок роста функции $p(k)$, а потом обоснуйте вашу гипотезу.

(*iv*) Получите оценку числа итераций модифицированного вероятностного алгоритма поиска минимального разреза, для получения заданной точности ε и приведите оценку общего числа операций.

Задача Д–14. $(0.01 + 0.03 + 0.01)$ **[Вероятностный алгоритм для языка ВЫПОЛНИМОСТЬ и дерандомизация].** Предположим, КНФ содержит m дизъюнктов и в каждый дизъюнкт входит ровно k литералов. Пусть X — случайная величина, равная числу выполненных дизъюнктов, если независимо и равновероятно приписать каждому литералу значения 0 или 1. Поскольку каждый дизъюнкт ложен лишь при одном значении литералов и матожидания суммируются, то математическое ожидание числа выполненных дизъюнктов равно: $E(X) = m(1-2^{-k})$.

Это значит, что существует логический набор, на котором выполнено не менее $E(X)$ дизъюнктов. Далее, как и в случае с задачами из $\mathcal{NP}$, возникает та же проблема. Мы знаем, что такой набор существует, но не знаем, как его найти, не используя полный перебор. В нашем случае, для нахождения искомого набора можно, во-первых, построить эффективный вероятностный алгоритм. И более того, можно провести дерандомизацию, т. е. конвертировать вероятностную процедуру в детерминированную (не сильно увеличивая сложность). Последнее возможно далеко не всегда, и одним из центральных вопросов теории сложности является соотношение между классами $\mathcal{P}$ и $\mathcal{BPP}$, т. е. верно ли, что всякую процедуру из $\mathcal{BPP}$ можно дерандомизировать?

Алгоритм нахождения набора заключается в следующем. Оценим вероятность события $p = Prob[X \geq m(1-2^{-k})]$ (т. е. что в КНФ *выполнено не менее* $m(1-2^{-k})$ *дизъюнктов*). Для этого распишем математическое ожидание: $E(X) = \sum_{i=1}^{m} i * Prob[X = i] \leq [m(1-2^{-k}) - 1](1-p) + mp$. Отсюда $p \geq (m2^{-k}+1)^{-1}$.

Такая оценка позволяет построить следующий вероятностный алгоритм для определения набора, выполняющего не менее $m(1-2^{-k})$ дизъюнктов: достаточно независимо повторить процедуру $\geq m2^{-k}+1$ раз и тогда с вероятностью $> \frac{1}{2}$ одна из попыток даст искомый набор.

(*i*) Будет ли предложенный алгоритм 1) Лас-Вегас алгоритмом; 2) Монте-Карло алгоритмом; 3) ни тем, ни другим?

Поскольку порождение случайного набора требует $O(n)$ операций, а проверка числа выполненных дизъюнктов требует $O(mk)$ операций, то одна итерация алгоритма занимает $O(m2^{-k}(mk+n))$

Теперь попробуем дерандомизировать процедуру.

Свяжем с каждым литералом x_i случайную величину Y_i, принимающую равновероятно значения 0 или 1, и будем считать $\{Y_i\}$, $i = 1,\dots,n$ независимыми. Значение величины Y_i будем обозначать маленькой буквой y_i. Мы используем т. н. метод *условных вероятностей*, который заключается в последовательном приписывании логических значений литералам так, чтобы на каждом шаге выполнялось неравенство: $E(X|y_1,\dots,y_j) \leq E(X|y_1,\dots,y_{j+1})$. Осуществить такой подход удается не всегда. В нашем случае ключевым является тождество[31]: $E(X|y_1,\dots,y_j) = \frac{1}{2}[E(X|y_1,\dots,y_j,Y_{j+1} = 1) + E(X|y_1,\dots,y_j,Y_{j+1}=0)]$. Таким образом, нужно научиться *детерминированно* приписывать литералу x_{j+1} значение, имеющее *большее условное математическое ожидание.* Для этого нужно разбить множество дизъюнктов на 4 непересекающиеся подмножества [1) уже выполненных; 2) не зависящих от x_{j+1}; 3) выполняющихся при $x_{j+1} = 1$; 4) выполняющихся при $x_{j+1} = 0$], вычислить условные мат. ожидания и присвоить литералу x_{j+1} значение, отвечающее большему мат. ожиданию.

[31] На самом деле, достаточно потребовать *квазивогнутости*: $E(X|\dots) \leq \max[E(X|\dots,Y_{j+1}=1), E(X|\dots,Y_{j+1}=0)]$.

(*ii*) Закончите вычисления, т. е. явно укажите, какие истинностные значения следует присваивать литералам. При вычислении условных вероятностей не забывайте о вкладе переменных, значения которым **ещё не присвоены**!

(*iii*) Проведите вычисления для 2-КНФ[32] $(\bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_6) \wedge (x_2 \vee \bar{x}_4) \wedge (\vee \bar{x}_5 \vee x_7) \wedge (\bar{x}_7 \vee x_8 \vee) \wedge (x_1 \vee \bar{x}_7)$

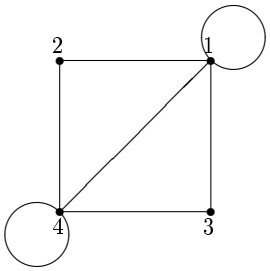
В заключение отметим, что по ходу рассуждений мы показали следующее утверждение (которое полезно доказать каким-то иным способом): *всякая k-КНФ, имеющая меньше 2^k дизъюнктов, выполнима.*

Кроме того, *для случая 3-КНФ, каждый дизъюнкт которой содержит ровно 3 литерала, мы построили $\frac{7}{8}$-приближенный (детерминированный и вероятностный) полиномиальный алгоритмы для NP-трудной задачи MAX-3-SAT, в которой требуется выполнить максимальное число дизъюнктов*[33]. И в этом бы не было ничего удивительного, если бы J.Håstad не показал (это довольно тяжело), что при $\mathcal{P} \neq \mathcal{NP}$ **никакая** эффективная процедура для задачи MAX-ВЫПОЛНИМОСТЬ не может давать большую точность.

Задание на седьмую неделю: 18.03–24.03. [0.1] Теоретико-числовые алгоритмы Литература: [Кормен 1, Глава 33] [Кормен 2, Глава 31], [ДПВ, Глава 2] [Виноградов]

Числовые алгоритмы[34]. 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. Вероятностные алгоритмы проверки простоты.

Задача 41. (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, т. е. разностному уравнению, и это подсказывает следующий *матричный* способ ее вычисления. Имеет место матричная формула[35] $\left(\begin{smallmatrix} g_n \\ g_{n+1} \end{smallmatrix}\right) = \left(\begin{smallmatrix} 0 & 1 \\ Q & P \end{smallmatrix}\right)^{n-1} \left(\begin{smallmatrix} g_1 \\ g_2 \end{smallmatrix}\right)$

При вычислении $\{g_n\}$ этим способом можно использовать быстрое, например, "индийское возведение в степень" n за $O(\log n)$ тактов[36].

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 трудоёмкость процедуры).

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

Оценим трудоёмкость 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–15. (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 трудоёмкость 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

[32] Контрольный вопрос: является ли указанная процедура полиномиальным алгоритмом для 2-КНФ?

[33] Это, значит, что алгоритм находит набор, на котором выполняется не менее $\frac{7}{8}$ от максимально возможного числа дизъюнктов, которые можно одновременно выполнить.

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

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

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

вычисления чисел $\{g_n\}$ по такому модулю. Проведите вычисления $A = g_{10000} \pmod{23}$. Оцените трудоёмкость вычисления $g_n \pmod{p}$ для этого случая.

В этой задаче вам предстоит разобраться, как использовать матричный алгоритм для вычислении рекуррентности 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 $\left(\begin{smallmatrix} 0 & 1 \\ Q & P \end{smallmatrix}\right)$ можно диагонализировать, т. е. привести ее к виду $S\left(\begin{smallmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{smallmatrix}\right)S^{-1}$, где λ_1, λ_2 — это собственные числа, а S — невырожденная матрица. Возводя в n-ю степень, получим $S\left(\begin{smallmatrix} \lambda_1^n & 0 \\ 0 & \lambda_2^n \end{smallmatrix}\right)S^{-1}$.

(*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[37]

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 основан на знаменитом *квадратичном законе взаимности*, о котором можно прочитать в книге **[Виноградов, гл. 2]** (и придумать алгоритм самостоятельно). Но есть и ещё 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 42. (*i*) Пусть $\text{НОД}(a, N) = 1$ и $a^{N-1} \neq 1 \pmod{N}$. Тогда по крайней мере для половины чисел из промежутка $1 \leq b < N$ выполнено $b^{N-1} \neq 1 \pmod{N}$.

Результаты этого простого, но важнейшего упражнения позволяют строить быстрые **вероятностные** алгоритмы типа Лас-Вегас, проверяющие простоту чисел. Ещё раз напомним, что это такое.

Речь идет о процедурах, которые, получив на вход n-битовую двоичную запись числа, могут быстро[38] 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**[39] 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 $\text{НОД}(a, N) > 1$ *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[40], 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[41] $\geq \frac{1}{2}$.

Комментарий (продолжение). Последняя задача должна навести на мысль, что вероятность нужно рассматривать как важный **вычислительный ресурс**, иногда позволяющий существенно снизить трудоёмкость. 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 счётчика 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

[37] 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, посвящённые аналогам теоремы Ферма для матриц, появились только в этом тысячелетии (см., www.mathnet.ru/mp238). В принципе, их мог написать сильный студент.

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

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

[40] 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]**.

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

достаточны хороши[42], 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)[43]. 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 [44].

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? Стандарный 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 вычёркиваем 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[45]. Грубо говоря, ничего лучше, чем рассмотренное выше 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.

Криптографии с открытым ключом. RSA

Начнём с модельного примера выбора секретного ключа с использованием публичных каналов обмена информацией. Могут ли Алиса и Боб[46] 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[47].

Рассмотрим также **СХЕМУ 2** выбора секретного ключа или схему обмена ключами Дифи и Хеллмана.[48] 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[49] $\pmod p$. 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 передаёт 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[50].

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 распространённой схеме шифрования с *открытым ключом* **RSA**[51]. 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 защищённые электронные подписи. Пусть открытый ключ Боба (e, n). Если он хочет электронно "подписать" своё 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

---

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

[43] 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/

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

[45] Следует отметить, что если изменить правила пересчёта вероятности на те, которые (экспериментально) выполняются в микромире, и рассмотреть т. н. квантовые алгоритмы, то задачу факторизации удаётся 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.

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

[47] 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.

[48] 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.

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

[50] Хотя к утверждению о надёжности нужно относится с известной долей скепсиса. Во всяком случае на сайтах, посвящённых 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 передёрнули. 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.

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

$(\mathrm{mod}\ n)$).

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 43. $(0.01 + 0.02)$ (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.

Показатели. Первообразные корни

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

Абелева мультипликативная группа $\mathbb{Z}_n^*$ (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, даёт ответ на вопрос, когда первообразные корни существуют.

Теорема 5 *В $\mathbb{Z}_n$ 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[52] $a^{\varphi(n)} = 1 \pmod n$ следует, что показатель всегда является делителем $\varphi(n)$.

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

Теорема 6 *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 $1, a, a^2, \dots, a^{\delta-1}$, 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 разобьём вычеты $0, 1, \dots, p-1$ 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 $\psi(p-1) = \varphi(p-1)$. □

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

Если в кольце $\mathbb{Z}_n$ 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 тяжёлыми задачами.

Задача 44. (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 45. (3×0.01) (i) Найдите все решения уравнения $\varphi(n) = 6$.
(ii) Найдите распределение вычетов по показателям $\mathbb{Z}_{19}$;
(iii) Найдите все первообразные корни $\pmod{19}$.

Задача Д–16. $(0.01 + 0.02 + 0.02)$ (i) Найдите все первообразные корни в $\mathbb{Z}_2$ и $\mathbb{Z}_4$.
(ii) Покажите, что $\mathbb{Z}_{2^k}^* \cong \mathbb{Z}_2 \times \mathbb{Z}_{2^{k-2}}$ для $k > 2$, и поэтому порядок любого элемента в группе $\mathbb{Z}_{2^k}^*$, $k > 2$ не больше $\frac{1}{2}\varphi(2^k)$ (поэтому в кольцах $\mathbb{Z}_8, \mathbb{Z}_{16}, \dots$ первообразных корней нет).
(iii) Пусть нечетный модуль n имеет два или более различных простых сомножителя, т. е. $n = n_1 n_2$, причём n — нечётное и $\mathrm{НОД}(n_1, n_2) = 1$. Покажите, что в $\mathbb{Z}_n$ первообразных корней нет. Аналогично проверяется, что первообразных корней нет по чётному модулю $2^k n$, где нечётное число n имеет не менее двух различных простых делителей.

Замечание. Для доказательства Теоремы 5 осталось убедиться в существовании первообразных корней по модулям p^k и $2p^k$ $k > 1$, p — нечетное простое число.

Решение линейных диофантовых уравнений и систем линейных сравнений. КТО
Литература: [Виноградов]

Задача 46. (3×0.01) Докажите или опровергните[53] существование полиномиальных алгоритмов для
(i) нахождения вычета, удовлетворяющего К[итайской] Т[еореме об] О[статках], т. е. нахождения единственного вычета, удовлетворяющего системе сравнений по различным простым модулям p_i, $i = 1, \dots, k$

$$\begin{aligned} x &= a_1 (\mathrm{mod}\ p_1) \\ &\dots \\ x &= a_k (\mathrm{mod}\ p_k) \\ 0 \leq\ x &< p_1 p_2 \dots p_k. \end{aligned}$$

[53] В предположении $\mathcal{P} \neq \mathcal{NP}$ или трудности задачи факторизации.

(*ii*) проверки совместности системы сравнений

$$\begin{aligned} x &= a_1 (\mathrm{mod}\ m_1) \\ &\dots \\ x &= a_k (\mathrm{mod}\ m_k) \end{aligned}$$

здесь модули m_i — произвольные натуральные числа;
(*iii*) решения системы линейных диофантовых уравнений $\sum_{j=1}^{m} a_{ij} x_j = b_i$, $a_{ij}, x_j, b_i \in \mathbb{Z}$, $i = 1, \dots, n$, $j = 1, \dots, m$.

Задание на восьмую неделю: 25.03–31.03. [0.2]

Хеширование
Литература: [Кормен 1, Глава 12] [Кормен 2, Глава 11], [ДПВ, §1.5]

Рекомендуем также посмотреть http://e-maxx/algo/string_hashes

Краткий конспект. Хеш- функции

Хеш- функции описывают специальную дисциплину обращения с большими массивами. Для построения хороших хеш- функций часто используют теоретико- числовые методы.

Мотивация для введения хеш-функций такова. Далее используется удачный пример из **[ДПВ]**.

Допустим, что мы хотим поддерживать динамический массив IP-адресов (скажем, клиентов или друзей в какой-то социальной сети). Напомним, что IP-адрес представляет 32-битный код, обычно разбитый на 8-битные куски $x_1.x_2.x_3.x_4$, например, 193.133.89.10. Понятно, что держать в памяти все 2^{32} возможных IP-адресов накладно, тем более, если, по вашим оценкам, число клиентов или друзей никак не сможет превысит порог, скажем, 250. Однако, если задать IP-адреса списком, то каждое обращение потребует просмотра всего списка, что может быть медленным. Поэтому предлагается дать каждому IP-адресу x "короткое имя" $h(x)$, быстро вычисляемое посредством специальной хеш-функции $h(\cdot)$, и ввести маленький массив из связанных списков, для их размещения. Например, в качестве короткого имени можно выбрать первый байт IP-адреса. При этом мы понимаем, что многие IP-адреса "склеятся", т. е. получат одно и то же короткое имя и будут помещены в связанный список под этим именем. Например, это произойдёт, если клиенты (или друзья) обитают в Азии. Также понятно, что как бы мы ни выбирали хеш-функцию, для нее априори всегда существуют совокупность "плохих" IP-адресов. Что же делать?

Вспомним, что неформально хеш-функция должна выдавать на выходе равномерно распределенные "случайные" имена, так чтобы длины связанных списков не слишком отличались. Мы, конечно, не можем разрушить возможную организацию во входах, но кто нам мешает ввести рандомизацию на множестве функций, применяемых для хеширования[54]? Например, можно действовать так. Зафиксируем простой модуль $n = 257$ близкий к 250. Выберем случайно четвёрку чисел $a = \{a_1, a_2, a_3, a_4\}$ по $(\mathrm{mod}\ 257)$ и определим хеш-функцию $h_a(x_1, x_2, x_3, x_4) = \sum_1^4 a_1 x_1 \ (\mathrm{mod}\ 257)$. Например, рассмотренная ранее хеш-функция, выделяющая старший байт равна $h_{1,0,0,0}$.

Пусть $x_1.x_2.x_3.x_4$ и $y_1.y_2.y_3.y_4$ — *различные* IP-адреса. Предположим, что коэффициенты a_i, $i = 1, 2, 3, 4$ выбираются случайно и независимо и равновероятно принимают значение из $\{0, 1, \dots, n-1\}$. Тогда[55]

$$Prob[h_a(x_1, x_2, x_3, x_4) = h_a(y_1, y_2, y_3, y_4)] = \frac{1}{n}. \qquad (1)$$

Иными словами, вероятность совпадения хеш-значений двух различных IP-адресов, вычисленных посредством случайно выбранной хеш-функции семейства, равна вероятности совпадения хеш-значений, если бы они выбирались случайно и независимо. В частности, если мы поместим m IP-адресов в хеш-таблицу, то к каждому имени в среднем будет "привешено" не более $\frac{m}{n}$ IP-адресов, чего мы и добивались.

Описанный принцип выбора хеш-функций называется **универсальным хешированием**.

Формальное определение таково. Семейство $\mathcal{H}$ хеш-функций, отображающих множество возможных ключей U в $\{0, 1, \dots, m-1\}$, называется *универсальным*, если для двух различных ключей $x, y \in U$ число функций $h \in \mathcal{H}$, для которых $h(x) = h(y)$, меньше или равно[56] $\frac{|\mathcal{H}|}{m}$. Из описанного примера следует, что семейство $\mathcal{H}_1 = \{h_a,\ a \in \{0, \dots, 256\}^4\}$ является универсальным

Анализ (**[Кормен 1, Теорема 12.3]** или гораздо более подробная **Кормен 2,Теорема 11.3]**) показывает, что универсальные семейства хеш-функций позволяют строить хеш-таблицы, с короткой в среднем длиной связанных списков.

Введём ещё одно понятие

Семейство $\mathcal{H}$ хеш-функций, отображающих множество возможных ключей U в $\{0, 1, \dots, m-1\}$, называется *k-универсальным*, если для любой последовательности k различных ключей $(x_1, \dots, x_k)$ случайный вектор $\langle h(x_1), \dots, h(x_k) \rangle$ (здесь h— случайная функция из $\mathcal{H}$) принимает равновероятно все m^k своих возможных значений.

Задача 47. (0.01) Докажите формулу (1).

Задача 48. (3×0.02) В этой задаче p — простое число, а все равенства понимаются по модулю p.
(*i*) Пусть $\vec{x}, \vec{y} \in (\mathbb{Z}/p\mathbb{Z})^n$. Пусть вектор $\vec{v} \in (\mathbb{Z}/p\mathbb{Z})^n$ выбирается случайно (каждый его элемент выбирается из $\mathbb{Z}/p\mathbb{Z}$ равновероятно и независимо от других). Найдите $\mathbb{P}(\langle \vec{v}, \vec{x} \rangle = \langle \vec{v}, \vec{y} \rangle)$.
(*ii*) Пусть $\vec{x}, \vec{y} \in (\mathbb{Z}/p\mathbb{Z})^n$. Пусть матрица $A \in (\mathbb{Z}/p\mathbb{Z})^{n \times m}$ выбирается случайно (каждый её элемент выбирается из $\mathbb{Z}/p\mathbb{Z}$ равновероятно и независимо от других). Найдите $\mathbb{P}(A\vec{x} = A\vec{y})$.
(*iii*) Постройте универсальное семейство $\mathcal{H}$ хеш-функций $h : (\mathbb{Z}/p\mathbb{Z})^n \to (\mathbb{Z}/p\mathbb{Z})^m$. Семейство должно иметь мощность $|\mathcal{H}| = p^{mn}$.

Задача Д–17. (0.02) Для хеширования множества из N элементов (элементы заранее не известны, но принимают значения из множества ключей $\gg N$) используют хеш-таблицу размера N^2 и хеш-функцию, случайно выбранную из универсального семейства. Докажите, что вероятность выбрать хеш-функцию удачно (избежать коллизий) не меньше $\frac{1}{2}$.

Стандартные структуры данных
Литература: [Кормен 1] [Кормен 2, Главы 10, 12], [Ш]

В теорминимум входят стандартные простые структуры данных: *массив, список (однонаправленный и двунаправленный), двоичное дерево, стек, очередь. очередь с приоритетами, куча (heap)*[57] и **вопросы, связанные с этими структурами могут быть включены в тест**. Более сложные структуры: *сбалансированное дерево (АВЛ- и/или $2-3$- и/ или красно-чёрное деревья)* мы рассмотрим в задании № 9, посвящённом сортировке. Амортизационный анализ будет востребован при изучению алгоритма

[54] Обратите, пожалуйста, внимание на этот прием: мы сознательно разрушаем возможную организацию входных данных, что позволяет нам в дальнейшем считать, что вход уже не содержит возможных "патологий", поскольку механизм разрушения зависимостей **не связан** с исходной организацией. Типичный пример: всем известный QUICKSORT.

[55] Полезно заглянуть **[Кормен 1, гл. 12], [Кормен 2, гл. 11]** (лучше посмотреть оба текста, поскольку они сильно отличаются).

[56] В **[Кормен 1]** требуется равенство.

[57] **Мы исходим из предположения, что этот раздел уже изучался вами на информатике.** В задании проводится качественный анализ этих структур по сложности.

манипуляции с деревьями UNION-FIND в задании № 10. Вы должны представлять, как реализовать такие структуры на компьютере, знать, какие операции над ними допускаются и уметь оценивать трудоёмкость этих операций. В дальнейшем при обсуждении конкретных алгоритмов (например, поиска-в-глубину или поиска минимального дерева в графе) мы затронем также вопросы сложности этих алгоритмов при их реализации посредством каких-то структур данных, причём вопрос о том, какую структуру следует выбрать является совсем непраздным. Конечно, в отдельных случаях может быть совершенно очевидно, что нужно использовать, но это бывает далеко не всегда.

Краткий конспект: стек, очередь, двоичная куча
автор: И. Козлов

Под структурой данных мы будем понимать некоторый абстрактный тип данных вместе с набором операций, которые составляют ее интерфейс. Операции позволяют добавлять, изменять, искать и удалять данные из структуры.

Программисты работают с абстрактными типами данных исключительно через их интерфейсы, поскольку реализация может в будущем измениться. Такой подход соответствует принципу инкапсуляции в объектно-ориентированном программировании. Сильной стороной этой методики является именно сокрытие реализации. Если пользователю доступен только интерфейс, то до тех пор, пока структура данных поддерживает этот интерфейс, все программы, работающие с этим абстрактным типом данных, будут функционировать корректно. Разработчики структур данных стараются, не меняя внешнего интерфейса и семантики функций, постепенно дорабатывать реализации, улучшая алгоритмы по скорости, надежности и используемой памяти.

Различие между абстрактными типами данных и структурами данных, которые реализуют абстрактные типы, можно пояснить на примере **стека** и **очереди**. Это простейшие структуры данных, реализующие динамические множества, которые поддерживают только операции вставки (*push* для стека и *enqueue* для очереди) и удаления (*pop* для стека и *dequeue* для очереди). Разница между ними в том, что стек представляет собой список элементов, организованный по принципу L[ast]I[n]F[irst]O[ut] "последним пришёл — первым вышел"), а очередь — F[irst]I[in]F[irst]O[ut]. Оба этих типа данных могут быть реализованы при помощи массива или линейного списка, с использованием различных методов динамического выделения памяти. Подробнее о деталях реализации можно посмотреть в **Кормен**. Однако каждая реализация определяет один и тот же набор функций, который должен работать одинаково (по результату, а не по скорости) для всех реализаций.

Задача 49. (2×0.01) (*i*) Покажите, как реализовать очередь с помощью двух стеков.
(*ii*) Покажите, как реализовать стек с помощью двух очередей.

Задача 50. (2×0.01) (*i*) Изобразите последовательность <13,4,8,19,5,11>, хранящуюся в дважды связанном списке, представленном с помощью нескольких массивов.
(*ii*) Выполните это же задание для представления с помощью одного массива.

Приведём список типичных операций над динамическим множеством S, элементами которого являются пары ключ-значение. Каждый конкретный интерфейс может реализовывать некоторые операции из этого списка или какие-то иные операции, как правило, связанные с нижеперечисленными.

SEARCH(S,k) Запрос возвращает указатель на элемент x заданного множества S, для которого $key[x] = k$ или NIL, если в множестве S такой элемент отсутствует.

INSERT(S,x) Пополняет заданное множество S элементом x.

DELETE(S,x) Удаляет из заданного множества S элемент x.

MINIMUM(S) Возвращает указатель на элемент множества S с наименьшим ключом. Аналогично формулируется запрос MAXIMUM(S).

SUCCESSOR(S,x) Возвращает указатель на элемент множества S, ключ которого является ближайшим соседом ключа элемента x и превышает его. Если же x — максимальный элемент множества S, то возвращается значение NIL. Аналогично формулируется запрос PREDECESSOR(S,x).

Задача Д–18. (4 × 0.01) Определите асимптотическое время выполнения в наихудшем случае перечисленных в таблице операций над элементами динамических множеств, если эти операции выполняются со списками перечисленных ниже типов (расшифровка обозначений в таблице: ULL (несортированный однократно связанный список); SLL (сортированный однократно связанный список); UDLL (несортированный дважды связанный список); SDLL (сортированный дважды связанный список).

	ULL	SLL	UDLL	SDLL
SEARCH(S,k)				
INSERT(S,x)				
MAXIMUM(S)				
SUCCESSOR(S,x)				

Очередь с приоритетом (англ. *priority queue*) — абстрактный тип данных в программировании, поддерживающий две обязательные операции— добавить элемент и извлечь максимум. Соответственно, интерфейсы, реализуемые очередью с приоритетом, следующие:

- insert(key, value) — добавляет пару (key, value) в хранилище;
- extract-minimum()— возвращает пару (key, value) с минимальным значением ключа, удаляя её из хранилища.

При этом меньшее значение ключа соответствует более высокому приоритету.

В некоторых случаях более естественен рост ключа вместе с приоритетом. Тогда второй метод можно назвать extract-maximum(). В качестве примера очереди с приоритетом можно рассмотреть список задач исполнителя. Когда он заканчивает одну задачу, он переходит к очередной — самой приоритетной (ключ будет величиной, обратной приоритету) — то есть выполняет операцию извлечения максимума. Начальник добавляет задачи в список, указывая их приоритет, то есть выполняет операцию добавления элемента. Очереди с приоритетом используются в некоторых эффективных алгоритмах на графах, например в алгоритме Дейкстры, а также в алгоритме пирамидальной сортировки.

Очередь с приоритетами может быть реализована на основе различных структур данных. Простейшие (и не очень эффективные) реализации могут использовать неупорядоченный или упорядоченный массив, связный список, подходящие для небольших очередей. При этом вычисления могут быть как "ленивыми" (тяжесть вычислений переносится на извлечение элемента), так и ранними (eager), когда вставка элемента сложнее его извлечения. То есть, одна из операций может быть произведена за время $O(1)$, а другая в худшем случае — за $O(N)$, где N — длина очереди. Более эффективными являются реализации на основе кучи, где обе операции можно производить в худшем случае за время $O(\log N)$. К ним относятся двоичная куча, биномиальная куча, фибоначчиева куча.

Рассмотрим далее двоичную кучу или пирамиду — такое двоичное дерево, для которого выполнены три условия:

- Значение в любой вершине не меньше, чем значения её потомков.
- Глубина листьев (расстояние до корня) отличается не более чем на 1 слой.
- Последний слой заполняется слева направо.

Такая пирамида называется невозрастающей (max-heap). Удобная структура данных для пирамиды — массив A, у которого первый элемент, $A[1]$ — элемент в корне, а потомками элемента $A[i]$ являются $A[2i]$ и $A[2i+1]$ (при нумерации элементов с первого). При нумерации элементов с нулевого, корневой элемент — $A[0]$, а потомки элемента $A[i]$ — $A[2i+1]$ и $A[2i+2]$. При таком способе хранения последние два условия выполнены автоматически.

Высота пирамиды определяется как высота двоичного дерева. То есть она равна количеству рёбер в самом длинном простом пути, соединяющем корень кучи с одним из её листьев. Высота кучи есть $\Theta(\log N)$. Если в пирамиде изменяется один из элементов, то она может перестать удовлетворять свойству упорядоченности. Для восстановления этого свойства служит процедура Heapify. Она восстанавливает свойство кучи в дереве, у которого левое и правое поддеревья удовлетворяют ему. Эта процедура принимает на вход массив элементов A и индекс i. Она восстанавливает свойство упорядоченности во всём поддереве, корнем которого является элемент $A[i]$.

Если i-й элемент больше, чем его сыновья, всё поддерево уже является пирамидой, и делать ничего не надо. В противном случае меняем местами i-й элемент с наибольшим из его сыновей, после чего выполняем Heapify для этого сына.

Процедура выполняется за время $O(\log N)$.

```
Heapify(A, i)
left = 2i
right = 2i+1
heap_size - количество элементов в куче
largest = i
if left <= A.heap_size и A[left] > A[largest]
  then largest = left
if right <= A.heap_size и A[right] > A[largest]
  then largest = right
if largest > i
  then Обменять A[i] , A[largest]
       Heapify(A, largest)
```

Задача 51. $(0.01 + 0.02)$ (i) Проиллюстрируйте работу процедуры $Heapify(A, 3)$ с массивом
$A =< 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 >$
(ii) Теперь построим пирамиду. Заметим, что если выполнить Heapify для всех элементов массива A, начиная с последнего и кончая первым, он станет пирамидой. В самом деле, легко доказать по индукции, что к моменту выполнения $Heapify(A, i)$ все поддеревья, чьи корни имеют индекс больше i, являются пирамидами, и, следовательно, после выполнения $Heapify(A, i)$ пирамидой будут все поддеревья, чьи корни имеют индекс, не меньший i.

Кроме того, $Heapify(A, i)$ не делает ничего, если $i > N/2$ (при нумерации с первого элемента), где N — количество элементов массива. В самом деле, у таких элементов нет потомков, следовательно, соответствующие поддеревья уже являются пирамидами, так как содержат всего один элемент.

Таким образом, достаточно вызвать Heapify для всех элементов массива A, начиная (при нумерации с первого элемента) с $[N/2]$-го и кончая первым. **(Упражнение. Вставьте, пожалуйста, нужные значки вместо вопросов в листинг ниже.)**

```
Build-Max-Heap(A)
  A.heap_size ? A.length
  for i ? [A.length/2] downto 1
    do Heapify(A, i)
```

Докажите, что хотя здесь происходит $n/2$ вызовов функции Heapify со сложностью $O(\log N)$, но время работы равно процедуры $Build - Max - Heap(A)$ равно $O(N)$.

Задача Д–19. (3×0.01) (i) Покажите, что пирамида реализует очередь с приоритетами. Для этого напишите функции $Heap - Extract - Max(A)$ и $Max - Heap - Insert(A, key)$. Докажите, что обе функции работают за $O(\log N)$.
(ii) Пирамиду можно построить с помощью многократного вызова процедуры Max-Heap-Insert для вставки элементов в пирамиду. Рассмотрим следующий вариант процедуры Build-Max-Heap:

```
Build-Max-Heap-By-Inserts(A)
  A.heap_size = 1
  for i = 2 to A.length
    do Max-Heap-Insert(A,A[i])
```

Всегда ли процедуры Build-Max-Heap и Build-Max-Heap-By-Inserts для одного и того же входного массива создают одну и ту же пирамиду? Докажите что это так или приведите контрпример.
(iii) Покажите, что в наихудшем случае для создания N-элементной пирамиды процедуре Build-Max-Heap-By-Inserts потребуется время $O(N \log N)$.

Амортизационный анализ
Литература: [Кормен 1, Главы 18, 22] [Кормен 2, Главы 17, 21], [АХУ, §§4.7–4.8]

Помимо измерений сложности конкретной процедуры вне контекста можно интересоваться её сложностью в длинной серии исполнений над постоянно меняющимися данными. В этой ситуации вполне может оказаться, что среднее время исполнения значительно меньше худшего времени исполнения. Поэтому имеет смысл рассмотреть так называемую *амортизированную* (или *учётную*) сложность алгоритма. Следующая стандартная задача поясняет и мотивирует эту постановку. Более сложные примеры мы разберём с следующем задании.

Задача 52. (0.01) Рассматривается тривиальная структура данных, которая хранит натуральное число n в двоичном представлении ($\lceil \log_2 n \rceil$ битов). При инициализации $n = 1$. Единственная поддерживаемая операция Add – увеличить хранимое число на 1. Ясно, что если число заканчивается k нулями, то эта операция потребует выполнения $(k + 1)$ элементарных действий. Поэтому оценка времени работы Add по худшему случаю – $\Theta(\log n)$. Представим теперь, что программист длительное время работает с этой структурой и совершает большое число N запросов к процедуре Add. Докажите, что суммарное время работы всех выполнений процедуры оценивается как $\Theta(N)$, а не $\Theta(N \log N)$. В таких ситуациях говорят, что *амортизированное* время работы операции Add есть $\Theta(1)$.

Задание на 9-ю неделю: 1.04–7.04. [0.12]
Умножение многочленов и алгоритм БПФ
Литература: [Кормен 1, гл.6 и §32] [Кормен 2, гл 6. и §30], [ДПВ] [Виноградов]

Нужно обязательно прочитать статью (она выложена на нашем сайте): P. Clifford, R. Clifford. Simple deterministic wildcard matching. Information Processing Letters 101 (2007) 53–54. и посмотреть сайт

https://en.wikipedia.org/wiki/Circulant_matrix

КЛЮЧЕВЫЕ СЛОВА (минимальный необходимый объем понятий и навыков по этому разделу): Дискретное преобразование Фурье (ДПФ); быстрое преобразование Фурье (БПФ); схемы БПФ, перемножение многочленов с помощью БПФ. Поиск подстрок посредством БПФ. Циркулянты. Решение линейных уравнений с циркулянтными матрицами с помощью БПФ. Системы линейных уравнений с теплицевыми и ганкелевыми матрицами.

Вы уже знакомы с парой процедур, которые до какой-то степени определяют лицо нашего предмета (и не только нашего!). Это,

конечно, алгоритм Евклида и метод Гаусса решения систем линейных уравнений. В этом задании мы разберём третий великий алгоритм: быстрое преобразование Фурье (БПФ). Несмотря на то, что этот метод был предложен значительно позже двух своих знаменитых собратьев[58], но в настоящее время он вряд ли уступает им по распространённости. Можно сказать, что в обыденной жизни вы встречаетесь с ним даже чаще, поскольку БПФ "вшито" чуть ли ни в каждый мобильник.

По определению, *дискретное преобразование Фурье (ДПФ)* — это отображение, переводящее последовательность коэффициентов (вообще говоря, комплексных) многочлена $f(x) = a_0 + a_1x^1 + \ldots + a_{n-1}x^{n-1} \in \mathbb{C}[x]$ в последовательность его значений в корнях n-й степени из единицы, т. е. $(a_0, a_1, \ldots, a_n) \mapsto (f(\omega_n^0), f(\omega_n^1), \ldots, f(\omega_n^{n-1}))$, где $\omega_n = \exp\frac{2\pi i}{n}$. На первый взгляд, кажется, что вычисление требует $\Omega(n^2)$ элементарных арифметических операций, ведь нужно подставить каждый корень из единицы в многочлен $f(\cdot)$ и повторить операцию n раз.

Обратите внимание, что мы пользуемся не нашей стандартной битовой моделью сложности, а используем "наивную" 'нью-мерическую' модель вычислений с плавающей запятой (float)". Иначе говоря, считаем, что у нас есть возможность достаточно точно производить ПРИБЛИЖЕННЫЕ операции с плавающей точкой так, что они существенно не повлияют на ответ. Сложностью называем полное число арифметических операций. Такая модель широко распространена на практике, но требует определенной квалификации вычислителя, причём, если степени полиномов и/или порядок используемых чисел достаточно велики, то уже обычным рецептом — повторением вычислений с двойной точностью — не отделаешься. Мы ещё вернёмся к этому вопросу.

Быстрое преобразование Фурье (БПФ) — это алгоритм, позволяющий вычислить ДПФ, используя $O(n \log n)$ операций.

Какое все это имеет отношение к мобильным телефонам? Приведём небольшой комментарий, хотя уверены, что вам все это известно на несколько порядков лучше, чем нам. Как известно, на заре эпохи вычислительных машин цифровые и аналоговые устройства серьёзно конкурировали, но в настоящее время остались лишь островки аналоговых вычислений[59]. Иначе говоря, любой непрерывный сигнал $\phi(t)$, будь то электромагнитный импульс, посылаемый вашим мобильником, давление в аэродинамической трубе, распределение цветов на экране вашего монитора и т.д. должен быть отцифрован (это, как все понимают, может быть очень даже нетривиальной задачей), т. е. представлен массивом коэффициентов его значений в достаточно большом количестве точек $f(t) \mapsto (a_0, a_1, \ldots, a_{n-1})$. Здесь t может быть временем, пространственной координатой и т.д. Предположим теперь, что моделируемое устройство, на которое поступает сигнал (модем, телефон, экран, система управления ракетой и т.д.) линейное[60] (грубо говоря, при изменении интенсивности вдвое воздействие также изменяется в два раза и выполняется принцип суперпозиции: реакция от суммы сигналов равна сумме реакций), тогда его поведение может быть описано как импульсная или весовая функция в технике, фундаментальное решение или функция Грина в математике и физике и пр. По определению, импульсная функция — это реакция системы на единичный импульс в нулевой момент $t = 0$, т. е. на δ-функцию. В нашей дискретной модели реакция тоже должна задаваться массивом $(b_0, b_1, \ldots, b_{n-1})$ (считаем, что после i тактов на выходе b_i, а через n тактов реакция затухает).

Итак, пусть в момент времени $T = 0$ на вход поступает сигнал с интенсивностью a_0, который преобразуется системой в $b_0 \cdot a_0$, т. е. отклик системы в момент $T = 0$ равен a_0b_0. Далее, в момент $T = 1$ на вход поступает сигнал с интенсивностью a_1, вызывающий реакцию b_0a_1, которую согласно принципу суперпозиции нужно просуммировать с *остаточным* воздействием сигнала, пришедшего в момент $T = 0$, т. е. a_0b_1, и отклик системы равен $a_1b_0 + a_0b_1$ и т.д. Таким образом, при наших предположениях о поведении системы получаем, что если $T < n$ (т. е. сигнал ещё поступает), то отклик равен $c_T = \sum_{i=0}^{T} a_ib_{T-i}$, а если $T \geq n$ (т. е. входной сигнал уже затух), то $c_T = \sum_{i=T-n+1}^{n-1} a_ib_{T-i}$. В момент $T = 2n-2$ отклик равен $c_{2n-2} = a_{n-1}b_{n-1}$, а далее отклик равен нулю. Таким образом, отклик системы $c_0, c_1, \ldots, c_{2n-2}$ в точности совпадает с последовательностью коэффициентов многочлена- произведения:

$$\sum_{i=0}^{2n-2} c_ix^i \stackrel{def}{=} \sum_{i=0}^{n-1} a_ix^i \sum_{i=0}^{n-1} b_ix^i.$$

Итак, если мы хотим эффективно обрабатывать сигналы, то нужно научиться быстро умножать полиномы. Обычный способ умножения "столбиком" требует порядка n^2 операций, а алгоритм БПФ позволяет умножать многочлены почти на порядок быстрее. Но сначала разберёмся, как, собственно, реализовать само это быстрое преобразование. Для этого мы используем симметрии корней из единицы. Заметим[61], что если n чётное, то квадраты всех корней из единицы степени n образуют $\frac{n}{2}$ корней из единицы степени $\frac{n}{2}$ (можете ли вы сказать, сколько корней степени n перейдёт в один корень степени $\frac{n}{2}$?). Теперь все готово для алгоритма. Попробуем применить технику "разделяй-и-властвуй". Итак, надо вычислить $f(\omega_n^k)$, $k = 0, \ldots, n-1$. Перепишем выражение по четным и нечётным индексам (считаем n четным) $f(\omega_n^k) = \sum_{i=0}^{\frac{n}{2}-1} a_{2i}\omega_n^{2ik} + \sum_{i=0}^{\frac{n}{2}-1} a_{2i+1}\omega_n^{2ik+k} = \sum_{i=0}^{\frac{n}{2}-1} a_{2i}(\omega_n^2)^{ik} + \omega_n^k \sum_{i=0}^{\frac{n}{2}-1} a_{2i+1}(\omega_n^2)^{ik}$, и задача разбивается на две аналогичные задачи для многочленов степени $\frac{n}{2} - 1$, поскольку ω_n^2 — это корень из единицы степени $\frac{n}{2}$. Получаем следующую рекуррентность для трудоёмкости алгоритма $T(n) = 2T(\frac{n}{2}) + O(n)$, откуда $T(n) = O(n \log n)$

Остаётся применить изложенную технику для умножения многочленов. Для этого нужно вспомнить, что произвольный многочлен степени $n-1$ может быть однозначно восстановлен по его значениям в произвольных n точках. Поэтому вместо того, чтобы пересчитывать коэффициенты многочлена $h(x) = f(x) \cdot g(x) = (\sum_{i=0}^{n-1} a_ix^i)(\sum_{i=0}^{n-1} b_ix^i) = \sum_{i=0}^{2n-2} c_ix^i = \sum_{i=0}^{2n-2}(\sum_{j=0}^{i} a_j \cdot b_{i-j})x^i$ (прямое вычисление коэффициентов требует $O(n^2)$ операций) мы сначала вычислим значения многочленов $f(x)$ и $g(x)$ в некоторых $2n-1$ точках интерполяции, получив массивы $F = \{f_i\}$ и $G = \{g_i\}$. Покомпонентно перемножая F и G, получаем массив $H = \{f_i \cdot g_i\}$, который, по построению, отвечает значениям многочлена-произведения $h(\cdot)$ в тех же точках интерполяции, и $h(\cdot)$ можно однозначно восстановить и получить ответ.

Итак, мы получили массив значений многочлена-произведения в корнях из единицы, и теперь нам нужно восстановить его коэффициенты. Конечно, если выбирать точки интерполяции совершенно произвольно, то не понятно, почему такой метод мог бы быть эффективнее прямого вычисления коэффициентов h. Однако, выбор точек интерполяции в корнях из единицы позволяет построить $O(n \log n)$-алгоритм перемножения многочленов! Точнее говоря, трудоемкость процедуры будет $O(n \log n \cdot M)$, где параметр M отвечает за трудоёмкость отдельных арифметических операций. Конечно, если считать, что точность фиксирована, то M — константа, связанная с конкретной реализацией арифметических операций с плавающей точкой, и такая запись не очень осмыслена. Значит нужно попробовать реализовать точные вычисления, считая, как обычно, коэффициенты многочленов целыми. (Такая постановка реализуется во многих системах символьных вычислений или, например, в алгоритмах быстрого перемножения больших чисел, которые, как мы помним, и используют точное БПФ в качестве подпрограммы.) Мы обсудим этот вопрос чуть ниже, изучая возможную реализацию ДПФ и БПФ в конечных полях и кольцах вычетов $\mathbb{Z}_n$.

[58] Алгоритм опубликовали в 1965 г. Д.Кули (James Cooley) и Д.Тьюки (John Tukey) — последний, между прочим, был консультантом по научным вопросам Джона Кеннеди,— но известно, что он применялся ранее и другими авторами. Как и следовало ожидать, его использовал "король математиков" К.-Ф. Гаусс в начале 19 века (записи были расшифрованы недавно).

[59] Хотя, с другой стороны, столь популярную в последнее время и даже проникшую в таблоиды модель квантовых вычислений, до которых мы, возможно, доберемся, можно рассматривать как явный реванш аналоговых вычислительных устройств. Если их удастся реализовать, то можно будет эффективно выполнять некоторые процедуры, весьма трудоёмкие для обычных компьютеров, например факторизовать числа.

[60] Это почти общее предположение. Если же система нелинейная, то рассматривают её линеаризацию.

[61] Хорошо бы действительно понять эти простые, но фундаментальные факты. Все они, конечно, доказываются в Кормене, но лучше попробовать получить их самому — это буквально одна строчка выкладок

А сейчас ответим на исходный вопрос, почему можно быстро за $O(n\log n)$ операций восстановить коэффициенты многочлена по его ДПФ. По определению, ДПФ задаётся умножением матрицы типа Вандермонда $\mathcal{F}_n = \|(\omega_n)^{ij}\|_{i,j=0,1,\ldots,n-1}$ на вектор-столбец $(a_0, a_1, \ldots, a_{n-1})$. Действительно, для корней из единицы справедливо тождество: для любого k выполнено

$$\sum_{i=0}^{n-1}(\omega_n^k)^i = \delta_{0k \bmod n}\cdot n \qquad (2)$$

(δ_{ij} — это символ Кронекера — дискретная δ-функция), поэтому обратную матрицу можно задать так:

$$\mathcal{F}_n^{-1} = \frac{1}{n}\|(\omega_n^{-1})^{ij}\|_{i,j=0,1,\ldots,n-1} \qquad (3)$$

и по аналогичной причине и обратное ДПФ (умножение матрицы $\mathcal{F}_n^{-1}$ на массив коэффициентов) можно вычислить, используя $O(n\log n)$ операций.

Ещё один момент, который хотелось бы затронуть в связи с БПФ,— это его возможное нерекурсивное исполнение. Оказывается, что БПФ можно задать явной схемой, которая к тому же допускает параллельное выполнение (см. рис. 1).

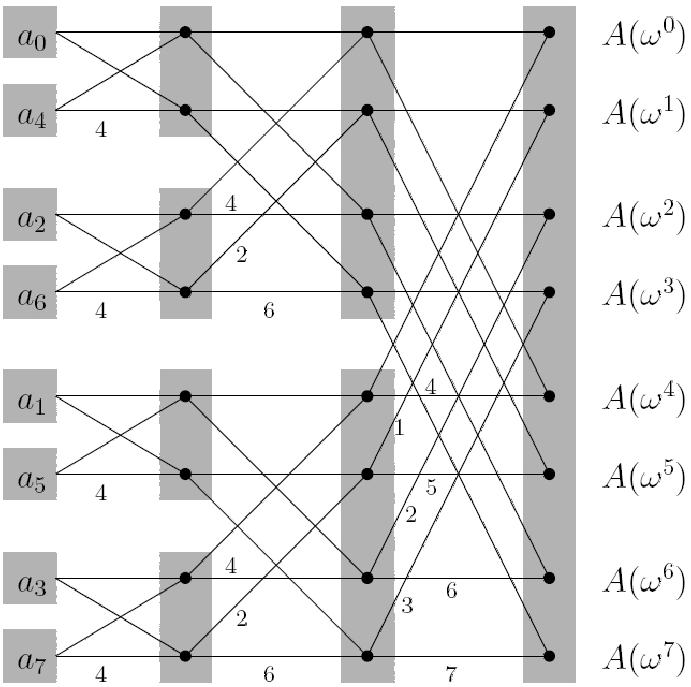


Рис. 1: Схема БПФ

На схеме рёбра-провода "несут" информацию слева направо, т. е. каждому ребру приписано (комплексное) число. Метка i на ребре означает, что число, приписанное проводу, нужно умножить на ω^i. В вершинах, обозначенных чёрными кружками, нужно просуммировать числа, приписанные проводам, приходящим *слева*. По схеме видно, что её основу составляет знаменитая "бабочка", отвечающая используемой в БПФ группировке степеней на чётные и нечётные и переходу к половинной размерности:

$$s_i = t_i + \omega^i u_i, \qquad s_{i+n/2} = t_i - \omega^i u_i.$$

Зачем нужно писать явные схемы, когда уже построен рекурсивный алгоритм? Во-первых, рекурсия может быть неэффективной при реализации, а явная схема проще. Во-вторых, схема показывает, как запустить алгоритм параллельно. В третьих, используя структуру схемы можно решать полезные смежные задачи, например, искать некоторые ошибки, см., задачу **№ 56**.

Другие приложения ДПФ и БПФ

1. БПФ и задача поиска образцов в тексте

Быстрое вычисление свёртки посредством БПФ используется во многих областях, например, в такой важной области, как анализ изображений. Изображение можно считать числовой матрицей [большого] размера, а типичной задачей может быть выделение [сравнительно небольшого] блока с заданными свойствами (под блоком понимается не минор, а подматрица исходной матрицы, в которой столбцы и строки идут подряд). Интересно, что содержательна и одномерная задача — поиск подстрок, с которой успешно справляется произвольный текстовый редактор. Казалось бы, эту задачу мы успешно решили в курсе ТРЯП, построив **оптимальный** линейный КМП-алгоритм. Но оказывается, что примерно в то же время (в 1974 г.), когда были предложены и КМП, и Boyer-Moore алгоритмы, и алгоритм Aho-Corasick для нескольких образцов, был предложен и быстрый алгоритм, основанный на БПФ. Его современная версия изложена ниже (см. также задачу **№ 54**). Трудоёмкость БПФ-алгоритма поиска подстрок отличается от оптимального логарифмическим фактором.

Идея БПФ-алгоритма поиска подстрок следующая. Нужно найти подстроку (образец) $p_0, \ldots, p_{m-1}$ в строке (тексте) $t_0, \ldots, t_{n-1}$, здесь p_i, t_j — это символы некоторого алфавита. Говорят, что подстрока входит с i-й позиции, если $p_j = t_{i+j}$, $j = 0, \ldots, m-1$. Если считать буквы алфавита различными целыми числами, то вхождение подстроки с i-й позиции эквивалентно обнулению суммы квадратов: $B_i = \sum_{j=0}^{m-1}(p_j - t_{i+j})^2 = \sum_{j=0}^{m-1}(p_j^2 - 2p_jt_{i+j} + t_{i+j}^2) = 0$, а вычисление массива чисел $\{B_i,\ i = 0, \ldots, n-m\}$ позволяет определить все места вхождения подстроки в текст.

"Наивный" алгоритм — прямой перебор требует $O(mn)$ операций. Но в курса ТРЯП изучался более быстрый, линейный, алгоритм. С помощью БПФ можно построить $O(n\log m)$-процедуру. Сумма $S = \sum\limits_{j=0}^{m-1} p_j^2$ присутствует как слагаемое в каждом B_i, и вычисляется за $O(n)$ шагов.

Рассмотрим два полинома степени не выше n: $T(x) = t_{n-1}x^{n-1} + \cdots + t_1x + t_0$, $P(x) = p_0x^{n-1} + \cdots + p_{m-1}$ Их произведение можно вычислить с помощью БПФ за $O(n\log n)$. Посмотрим на коэффициент их произведения $C(x)$ при x^{m-1+i} $(0 \le i \le n-m)$:

$$c_{m-1+i} = p_0t_i + p_1t_{i+1} + p_2t_{i+2} + \cdots + p_{m-2}t_{m-2+i} + p_{m-1}t_{m-1+i} = \sum_{j=0}^{m-1} p_jt_{j+i}$$

Как видим, это одно из слагаемых для B_i. Таким образом, алгоритм для подсчета всех B_i таков. Сначала вычисляем за $O(n\log n)$ шагов коэффициенты произведения $C(x)$ указаных многочленов. Далее за максимум $O(n)$ шагов вычисляем сумму квадратов S и за $O(n)$ шагов считаем сумму первых m квадратов t_i, т. е. $H = \sum\limits_{j=0}^{m-1} t_j^2$. Далее вычисляем $B_0 = S - 2\cdot c_{m-1} + H$ и $B_1 = S - 2\cdot c_m + \sum\limits_{j=0}^{m-1} t_{j+1}^2 = B_1 + 2\cdot c_{m-1} - 2\cdot c_m - t_1^2 + t_m^2$.

Для этого нам потребуется $O(1)$ операций. Аналогично получим $B_i = B_{i-1} + 2(c_{m-2+i} - c_{m-1+i}) - t_{i-1}^2 + t_{m-1+i}^2$, т. е. для получения каждого следующего члена требуется $O(1)$ операций. Таким образом, для вычисления всех членов потребуется еще $O(n)$ операций. Таким образом, весь алгоритм асимптотически требует $O(n\log n)$ операций.

Ту же идею для проверки вхождения подстроки можно использовать, если разрешается использовать символ джокера (в курсе ТРЯП он обозначался знаком (метасимволом) «?»). Тогда в тех же обозначениях нужно вычислить массив $\{A_i,\ i = 0, \ldots, n-m\}$, где $A_i = \sum_{j=0}^{m-1}(p_j^3t_{i+j} - 2p_j^2t_{i+j}^2 + p_jt_{i+j}^3)$.

Важное замечание. Как уже отмечалось трудоёмкость БПФ-алгоритмов поиска подстрок отличается от оптимальных комбинаторных логарифмическим фактором. Но, если допустить, что в текст или в образцы могут входить "джокеры" (wildcard characters), то БПФ-алгоритмы становятся рекордными. А сама **проблема существования линейных алгоритмов поиска образцов с "джокерами" открыта более сорока лет.**

1. Циркулянты

Ещё одно важное приложение ДПФ связано с эффективным решением специальных систем линейных уравнений, в которых матрицы коэффициентов имеют специальные симметрии, а именно,

постоянны вдоль каких-то "диагоналей" и тогда умножение на такую матрицу можно представить свёрткой. В качестве примера таких матриц обычно рассматриваются циркулянты (строки получены циклическим сдвигом фиксированного вектора), теплицевы или ганкелевы матрицы, у которых на каждой диагонали, параллельной главной (соответственно, параллельной побочной), стоят (свои) равные элементы.

Вы должны знать, как, используя БПФ, решать системы линейных уравнений порядка $n\times n$ с циркулянтной матрицей, используя $O(n\log n)$ операций, и представлять себе, как выполнить эту задачу для случая теплицевых или ганкелевых матриц, используя $O(n^2)$ операций.

Задача 53. $(0.02+0.02)$. (i) Найдите произведение многочленов $A(x)=3x+2$ и $B(x)=x^2+1$, используя рекурсивный $O(n\log n)$-алгоритм БПФ.

(ii) Вычислите **обратное ДПФ** массива (I — мнимая единица) $A=[10, 3I\sqrt{2}+2+2I, 0, 3I\sqrt{2}+2-2I, -2, -3I\sqrt{2}+2+2I, 0, -3I\sqrt{2}+2-2I]$, используя схему БПФ для $n=8$ на рис. 1. Возможно, вычисление будет удобнее проводить символически для $\omega=\exp\left(\frac{\pi I}{4}\right)$.

Задача 54. $(0.01+0.02)$. (i) Постройте $O(n\log n)$-БПФ-алгоритм для поиска подстрок в тексте с "джокерами".
(ii) Покажите, как понизить трудоёмкость вашей процедуры до $O(n\log m)$.

Задача 55. (0.01). Используя ДПФ, найдите решение системы линейных уравнений $Cx=b$. где C — это циркулянтная матрица, порождённая вектором столбцом $(1,2,4,8)^t$, а $b^t=(16,8,4,2)$.

Задача Д–20. (0.03). Дан множество различных чисел $A\subseteq\{1,\dots,m\}$. Рассмотрим множество $A+A$, образованное суммами элементов A. Докажите или опровергните существование процедур построения $A+A$, имеющих субквадратичную трудоемкость $o(m^2)$.

Задача 56. (0.02). (**[Кормен 1, упр. 32.3-4] или [Кормен 2, упр. 30.3-4]**. Предположим, что в схеме БПФ, изображённой на рис. 1, вышел из строя ровно один сумматор, причём он выдаёт число 0 независимо от входа. Сколько и каких последовательностей нужно подать на вход, чтобы идентифицировать дефектный сумматор?

ДПФ и БПФ в кольце вычетов $\mathbb{Z}_n$

Сначала мы рассмотрим простой случай, когда кольцо — это простое поле GF_p и попробуем просто осуществить в нем ДПФ. Понятно, что роль корня из единицы должен играть первообразный корень, поскольку мы знаем, что определитель Вандермонда, составленный из степеней произвольного первоообразного корня, обладает всеми нужными нам свойствами, причем в арифметике $\mathbb{Z}_p$, потому что речь идёт опять о сумме конечной геометрической прогрессии, период которой делится на порядок данного элемента. Итак для того, чтобы перемножить многочлены можно взять достаточно большое (насколько большое?) простое число p, перемножить многочлены $\pmod p$, используя ДПФ в конечном поле, а затем восстановить коэффициенты.

Задача 57. (0.03). Выберите подходящее простое число p и перемножьте многочлены $A(x)$ и $B(x)$ над конечным полем, а затем восстановите многочлен-произведение над $\mathbb{Z}$. Обоснуйте выбор p. Скажем, можно ли взять $p=5$? Или следует выбрать $p=7$ или больше?

В предыдущей задаче мы обосновали ДПФ над конечным полем, но можно ли провести трюк с БПФ? Об этом следующие задачи. Напоминаем, что, на самом деле, мы пытаемся уточнить предыдущую модель, когда предполагалось, что арифметические операции проводятся точно. И обычно учебники по численным алгоритмам этим и ограничиваются, а все трудности, связанные с априорной конечной точностью вычислений выносят в другие разделы (и под другими названиями типа "устойчивости", "обусловленности" и т. д.)[62].

Предположим, что в $\mathbb{Z}_n$ есть примитивный корень ξ степени[63] 2^k. Неформально модуль n будет означать максимальную границу модулей коэффициентов, которые могут встретиться в сомножителях и в произведении, а степень произведения не должна превышать $l=2^k-1$. Степень двойки требуется для того, чтобы прошла рекурсия в алгоритме БПФ

Задача Д–21. $(0.01+0.01+3\times 0.02)$.
(i) Покажите, что 3 является примитивным корнем 8-й степени в простом поле $\mathbb{Z}_{41}$.

Рассмотрим многочлен $h\in\mathbb{Z}_n[x]$ степени не выше $l=2^k-1$. Матрицу $\Xi=\|\xi^{ij}\|_{i,j=0,1,\dots,l}$ назовем матрицей ДПФ. По определению, ДПФ многочлена h равно произведению матрицы Ξ на вектор-столбец коэффициентов h

(ii) Покажите, что для любого $j=0,1,\dots,k-1$ элемент ξ^{2^j} — есть примитивный корень степени 2^{k-j} в $\mathbb{Z}_n$.

Из последнего упражнения вытекает, что можно быстро вычислять ДПФ произвольного многочлена h степени не выше l в $\mathbb{Z}_n$ ровно таким же рекурсивным алгоритмом, который мы использовали в $\mathbb{C}$.

(iii) Считая, что многочлены A и B (из задания) заданы над $\mathbb{Z}_{41}$ и $\xi=3$ вычислите рекурсивным алгоритмом их ДПФ.

Перемножая покомпонентно вычисленные в предыдущем пункте ДПФ многочленов A и B, мы получаем ДПФ произведения C. Теперь нам осталось проверить, что можно эффективно восстановить коэффициенты C по его ДПФ. Для этого нужно проверить выполнения равенства, аналогичного (3).

(iv) Докажите, что $\Xi^{-1}=(l+1)^{-1}\|(\xi^{-1})^{ij}\|_{i,j=0,1,\dots,l}$. Формула, конечно, справедлива для произвольной степени примитивного элемента ξ, а не только для степени 2^k, как в нашей спецификации.

(v) Используя результаты (iii) и (iv), вычислите ДПФ многочлена C и восстановите его коэффициенты рекурсивным алгоритмом БПФ [умножения обратной матрицы Ξ^{-1} на вектор столбец ДПФ].

Конечно, выбранный нами размер поля может оказаться недостаточным для того, чтобы произвести вычисления, поэтому дополнительно приведите обоснование корректности (или некорректности) выбора размера поля (само вычисление должно быть выполнено в любом случае).

Последний вопрос, который хотелось затронуть в этой теме, касается реализации БПФ для случая, когда порядок n не равен степени двойки. Как мы видели, в некоторых случаях, например, для умножения полиномов можно добавить необходимое количество нулевых коэффициентов, но хотелось бы иметь какой-то универсальный рецепт. Такие способы действительно есть. Мы крат-

[62] Понятно, почему так происходит: анализировать алгоритмы в битовой модели обычно значительно сложнее. Кроме того, довлеет ещё и традиция.

[63] Это, по определению, значит, что $\xi^i\neq 1$, $i=0,1,\dots 2^k-1$, а $\xi^{2^k}=1$.

ко изложим один из них, т. н. алгоритм L. Blueshtein (1968)[64]: "Пусть элементы матрицы $\mathcal{F}_n$ нумеруются индексами от 0 до $n-1$. В позиции (k,l) находится число $\omega^{kl} = \omega^{(k^2+l^2-(k-l)^2)/2} = \omega^{k^2/2}\omega^{-(k-l)^2}\omega^{l^2/2}$.

Поэтому матрица Фурье расщепляется в произведение трёх матриц ($0 \le k,l \le n-1$):

$$\mathcal{F}_n = DAD, D = \begin{bmatrix} \omega^{0^2/2} & & \\ & \ddots & \\ & & \omega^{(n-1)^2/2} \end{bmatrix}, A = \left[\omega^{-(k-l)^2/2}\right]$$

Таким образом, умножение на матрицу Фурье произвольного порядка n сводится к умножению на тёплицеву матрицу A того же порядка n. Последнее сводится к умножению на циркулянтную матрицу порядка $n \le N = 2^L < 4n$.

В итоге все сводится к троекратному применению алгоритма быстрого преобразования Фурье специально выбранного порядка $N = 2^L$. Описанная возможность получения быстрого преобразования Фурье без ограничений на его порядок является, вероятно, самой простой, но не единственной и не всегда наилучшей для практических вычислений."

Задание на 10-ю неделю: 8.04–14.04. [0.18] Сортировка. Раздел 9 программы Литература: [Кормен 1], [Кормен 2, гл 2], [ДПВ]

Разрешающие деревья. Нижние оценки сортировки

Обсудим вопрос о минимальном числе T_{min} попарных сравнений, необходимых для нахождения минимального из n чисел. Для этой задачи алгоритм очевиден: нужно последовательно сравнивать числа, оставляя при каждом сравнении минимальное. Возникает правдоподобная гипотеза, что $T_{min} = n-1$. Заметим, что даже в столь простой задаче ответ не очевиден, в частности, не проходит традиционный аргумент "по размеру входа", поскольку в $\frac{n}{2}$ сравнениях могут участвовать все числа, и речь фактически идет о том, какую часть информации о числах можно при сравнении передать. Рассмотрим два подхода к получению нижних оценок подобного рода.

Первый подход связан с понятием разрешающего дерева для алгоритмов сортировки **[Кормен 1 §9.1], [Кормен 2 §8.1]**. Напомним, что произвольный алгоритм A сортировки массива из n чисел $\{a_1,\dots,a_n\}$ посредством попарных сравнений можно следующим образом изобразить в виде корневого двоичного дерева D_A. Каждая внутренняя вершина v дерева помечена некоторым сравнением $a_i?a_j$, а в паре выходящих из v ребер одно ребро имеет пометку $\le$, а другое $\ge$. Листья D_A помечены соответствующими перестановками $\{\pi_1,\dots,\pi_n\}$, которые упорядочивают массив. Каждому конкретному входу $\{a_1,\dots,a_n\}$ отвечает его **реализация** — путь от корня к листу в D_A.

Совершенно аналогично даётся определение разрешающего дерева для задачи поиска минимального элемента, поиска медианы и т. д. (все сводится к изменению пометок листьев). Мы сохраним для этих «специализированных» деревьев обозначение D_A. На языке разрешающих деревьев утверждение о том, что $T_{min} = n-1$, эквивалентно следующему: в любом корректном алгоритме поиска минимального элемента в массиве из n чисел, использующем только попарные сравнения, каждый реализуемый путь от корня к листу имеет не менее $(n-1)$-го ребра.

Назовем это **утверждением** $\mathcal{A}$.

Произвольному корректному алгоритму A нахождения минимума попарными сравнениями и произвольному реализуемому пути P в разрешающем дереве D_A отвечает (неориентированный) граф $G_A^P = (V,E)$ на n вершинах, в котором есть ребро $(v_i,v_j) \in E$, если и только если в пути P какая-то вершина имеет пометку $a_i?a_j$.

Утверждение $\mathcal{B}$. Для корректности алгоритма A нахождения минимума необходимо, чтобы граф G_A^P был связен.

[64] Далее мы приведём отрывок из книги Тыртышников Е.Е. "Матричный анализ и линейная алгебра", где он описан, правда без ссылок на источники.

Задача 58. $(2 \times 0.01 + 0.02)$ (i) Докажите импликацию: $\mathcal{B} \Rightarrow \mathcal{A}$.

(ii) Докажите утверждение $\mathcal{B}$.

Теперь дадим другое доказательство этой нижней оценки. Отметим, что само доказательство будет иллюстрацией методов **амортизационного анализа** для получения нижних оценок.

Для этого запишем шаги алгоритма в формате конфигураций (a,b,c,d), где a элементов пока не сравнивались, b элементов были больше во всех сравнениях, c элементов были меньше во всех сравнениях, d были и больше, и меньше в сравнениях, т. е. начальная конфигурация такова: $Init = (n,0,0,0)$. Введем "потенциальную функцию", определённую на конфигурациях: $f[(a,b,c,d)] = a+c$. Мы оценим трудоёмкость алгоритма, просто поделив "разность потенциалов" между начальной и конечной конфигурациями на максимальное изменения потенциала за один шаг алгоритма.

(iii) Покажите, что при любом сравнении потенциал $f(\cdot)$ может уменьшиться не больше, чем на единицу, и что отсюда вытекает, что число шагов любого такого алгоритма не меньше $n-1$.

Покажем, что любой алгоритм нахождения медианы массива из n элементов посредством попарных сравнений имеет сложность $T(n) = \frac{3n}{2} - O(\log n)$.

Задача 59. $(0.02 + 0.03)$ (i) Покажите, что любое разрешающее дерево поиска медианы позволяет также восстановить индексы всех элементов, бóльших медианы, и всех элементов, мéньших медианы.

Из этой задачи вытекает, что нахождение медианы эквивалентно с виду более сложной задаче: найти медиану и массив L элементов, бóльших ее $(\frac{n}{2}-1)$ элементов.

(ii) Покажите, что любое разрешающее дерево для медианы содержит путь от корня к листу длины $\frac{3n}{2} - O(\log n)$.

Комментарий. Можно использовать два соображения. Во-первых, если из дерева T для медианы выкинуть все сравнения, в которых участвуют элементы L, то получится дерево T_L поиска максимума (в нем максимум — это медиана). А из предыдущей задачи следует, что T_L должно иметь $\ge 2^{\frac{n}{2}-1}$ листьев. Во-вторых, массив L может быть произвольным, а отсюда можно получить оценку снизу на число листьев (и на высоту) T.

Наилучшие известные современные оценки: $(2+\varepsilon)n \le T(n) \le 2.95n$.

Задача Д–22. (0.03) **[Кормен 1, задача 10-1-2]** Рассмотрим стандартную рекурсивную процедуру одновременного поиска максимума и минимума **[Кормен 1, §10.1]**. Покажите, используя подходящую потенциальную функцию, что этот алгоритм является оптимальным по числу использованных сравнений.

Комментарий. Здесь начальная и конечная конфигурации таковы: $Init = (n,0,0,0)$, $Final = (0,1,1,n-2)$. Нужно показать, что необходимо не менее $k = \lceil \frac{3n}{2} \rceil - 2$ сравнений. В тексте можно использовать только аргументы, относящиеся к потенциальной функции. Нельзя апеллировать к авторскому представлению о том, что какие-то действия "неоптимальные". Формат ответа, как и для рассмотренного выше выбора минимального элемента, должен быть таков.

1. Потенциальную функцию $f(\cdot) = f(a,b,c,d)$ следует указать явно.
2. При любом сравнении значения $f(\cdot)$ не могут уменьшиться больше, чем на некоторую величину δ (скажем, единицу).
3. $f(Init) - k\delta = f(Final)$.
4. На самом деле, легко проверить, что в классе линейных функций такая потенциальная функция не существует, поэтому

нужно либо усложнить вид функции, либо считать, что функция определена только на части входов.

Анализ сложности алгоритма Quicksort

Рассмотрим алгоритм быстрой сортировки с каким-нибудь детерминированным выбором "барьерного элемента". Обозначим через $\mathrm{t}(A_n)$ время работы алгоритма на массиве A_n длины n. По определению, средним временем работы алгоритма называется величина

$$\mathbb{E}\, t(n) = \frac{1}{n!} \sum_{A_n} t(A_n). \qquad (4)$$

На эту формулу можно взглянуть и немного по-другому. *Если считать равновероятными все $n!$ возможных способов упорядочения входного массива длины n*, то среднее время работы алгоритма, заданное формулой выше,— это, опять по определению, *математическое ожидание* времени работы алгоритма. Определим рекурсивную процедуру сортировки типа Quicksort, которую мы назовём Permsort следующим образом. *Сначала алгоритм случайным образом переставляет элементы текущего массива, причём таким образом, чтобы все возможные перестановки были равновероятны*[65], *а потом сортирует массив, выбирая барьерный элемент каким-то стандартным способом, например, используя самый правый.*

Задача 60. (2×0.02) (i) Покажите, что для среднего времени работы алгоритма Permsort справедливо рекуррентное соотношение.

$$\mathbb{E}\mathrm{t}(n) = \Theta(n) + \frac{1}{n}(\mathbb{E}\mathrm{t}(0) + \mathbb{E}\mathrm{t}(n-1)) + \frac{1}{n}(\mathbb{E}\mathrm{t}(1) + \mathbb{E}\mathrm{t}(n-2)) + \ldots$$
$$+ \frac{1}{n}(\mathbb{E}\mathrm{t}(n-2) + \mathbb{E}\mathrm{t}(1)) + \frac{1}{n}(\mathbb{E}\mathrm{t}(n-1) + \mathbb{E}\mathrm{t}(0)) =$$
$$= \Theta(n) + \frac{2}{n}(\mathbb{E}\mathrm{t}(1) + \mathbb{E}\mathrm{t}(1) + \ldots + \mathbb{E}\mathrm{t}(n-1)). \qquad (5)$$

Комментарий. (Возможный вариант решения.) Считаем, что время работы алгоритма на конкретном входе пропорционально числу проведённых сравнений[66]. *Нужно понять, что, по определению, в* (4) *записана сумма средних значений, т. е. сумма сумм. Теперь формулу* (5) *можно получить, изменяя порядок суммирования.*

(ii) Используя эту рекурренту, покажите, что $\mathbb{E}\mathrm{t}(n) = \Theta(n \log n)$.

В стандартном алгоритме Quicksort с рандомизацией при каждом (рекурсивном) обращении барьерный элемент выбирается случайным образом, причём равновероятно. Во всех изданиях **[Кормен]** проводится подробный анализ среднего времени работы алгоритма, но в **[Кормен 1]** описан специальный приём, который и будет проанализирован в следующей задаче.

Переопределим выбор случайного барьерного элемента следующим образом Будем считать, что изначально каждому элементу приписывается **ранг** — целое число от 1 до n. А выбор барьерного элемента будем проводить, выбирая каждый раз из текущего массива элемент "с минимальным рангом"[67].

Приведём цитату из **[Кормен 1]**: "Можно проверить, что это равносильно независимым выборам элементов на каждом шаге: *на первом шаге каждый из элементов может быть выбран с равной вероятностью, после такого выбора в каждой из групп все элементы также равновероятны и т.д.*"

Задача 61. (0.02) Обоснуйте эквивалентность обоих способов выбора, т. е. проверьте корректность утверждения выделенного курсивом в цитате выше.

Используя "ранги", легко вывести, что в алгоритме Quicksort элементы, которые после упорядочения попадают на i-е и j-е место, соответственно ($i \neq j$), сравниваются в $p_{ij} = \frac{2}{|i-j|+1}$-й доле случаев, что сразу дает оценку для трудоемкости в среднем: $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p_{ij} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = O(n \log n)$.

Задача 62. (0.01) Покажите, что если в любой модификации алгоритме Quicksort в качестве барьерного элемента использовать медиану текущего массива, причём искать её посредством стандартного линейного алгоритма, то его сложность по наихудшему случаю станет $O(n \log n)$.

Задача Д–23. ($0.03 + 0.03$) (i) Дано n ключей и n замков. Все ключи и все замки различны между собой, а каждый ключ подходит к единственному замку. Ключи (и замочные скважины) упорядочены по величине, но визуально отличия неразличимы. На каждом шаге можно попытаться вставить конкретный ключ в конкретный замок и заключить, что он подходит или больше, или меньше искомого. Постройте вероятностный алгоритм подбора ключей, требующий в среднем $o(n^2)$ шагов.

Комментарий. Очевидно, что прямой перебор подходящих пар ключей и замков требует квадратичного числа шагов. Удивительным кажется то, что для этой задачи построен детерминированный $o(n^2)$-алгоритм. Он очень хитрый.

(ii) Покажите, что любая детерминированная процедура подбора ключей требует $\Omega(n \log n)$ шагов.

Структуры данных для динамической сортировки
Манипуляции с деревьями

Литература: **[Кормен 1, гл. 18–19] или [Кормен 2, гл. 17–18], [АХУ], [Ш]**.

Изучим несколько структур данных древесного типа, которые позволяют эффективно динамически обращаться с упорядоченными массивами (т. е. нужно динамически поддерживать упорядоченный массив, считая, что элементы могут, например, добавляться и/или удаляться и пр.). Впервые подобную структуру данных, позволяющую выполнять удаления/вставки в n элементном массиве за $O(\log n)$ операций, предложили в 1963 году Г. Адельсон-Вельский и Е. Ландис[68]. Мы начнем изучение этой темы и рассмотрим несколько структур данных, позволяющих эффективно поддерживать динамические операции. При анализе трудоёмкости опять очень полезным оказывается метод т. н. *амортизационного анализа*, т. е. вычисление трудоёмкости не отдельной, а сразу целой серии операций.

Пусть x — двоичное дерева поиска[69]; обозначим $size[x]$ число ключей в поддереве с вершиной x. Выделим для $size[\cdot]$ поле в каждой вершине дерева. Пусть α — число и $1/2 \leq \alpha < 1$. Будем говорить, что вершина x дерева, не являющаяся листом, *α-сбалансирована*, если $size[[left[x]] \leq \alpha size[x]$ и $size[[right[x]] \leq$

[65] О том, что под этим выражением понимается формально, и как реализовать такую процедуру за линейное время, написано в **[Кормен 2, §5.3]**.

[66] В изданиях **[Кормен]** выше второго аналогичный факт явно формулируется и доказывается для стандартного алгоритма Quicksort.

[67] Трюк этот совсем нетривиальный: вместо того, чтобы каждый раз вызывать подпрограмму RANDOM, вы генерируете случайную перестановку и в дальнейшем используете только её.

[68] Отсюда и название самой структуры по первым буквам фамилий авторов — АВЛ-дерево.

[69] Что это такое?

$\alpha size[x]$ ($left$ и $right$ — это указатели на левого и, соответственно, правого потомка x в двоичном дереве). Дерево называется *α-сбалансированным*, если все его внутренние вершины α-сбалансированы.

Задача 63. $(0.02 + 0.02)$ (i) Покажите из определения, что для *любой* вершины x 1/2-сбалансированного дерева выполнено:

$$size[[left[x]] - size[[right[x]] \in \{-1, 0, +1\}$$

.

(ii) **[Кормен 1, задача 18.3 (б)]или [Кормен 2, задача17.3 (б)]** Покажите, что поиск элемента в α-сбалансированном двоичном дереве с n вершинами выполняется за $O(\log n)$.

В следующей задаче будет показано, что α-сбалансированные деревья можно эффективно динамически балансировать, т. е. осуществлять вставку и удалению можно за *учётное время* $O(\log n)$. О том, что такое учётное время тоже немного говорилось и ранее, но сейчас необходимо обязательно прочитать в **Кормене** начало главы 18 (I) (или 17 (II)).

Мы снова используем *метод потенциалов*.

Задача Д–24. $(0.02 + 0.01 + 0.03 + 0.03)$. **[Кормен 1, задача 18.3 (б)]или [Кормен 2, задача17.3 (б)]**
(i) Пусть x — вершина двоичного дерева поиска. Постройте алгоритм, использующий время $\Theta(size[x])$ и дополнительную память $O(size[x])$, для преобразования поддерева с корнем x в 1/2-сбалансированное дерево.

Далее считаем, что $\alpha > 1/2$, и после выполненных стандартным способом[70] операций удаления или вставки, следующим образом производится балансировка: выбирается самая высокая вершина результирующего дерева, которая перестала быть α-сбалансированной и все ее корневое поддерево перестраивается в 1/2-сбалансированное посредством алгоритма из пункта (i).

Используем метод потенциалов с потенциальной функцией:

$$\Phi(T) = c \sum_{x \in T\,:\,|\Delta(x)| \geq 2} |\Delta(x)|,$$

здесь T — двоичное дерево поиска, а $c > 0$ — достаточно большая константа, зависящая от α.

(ii) Покажите, что потенциал 1/2-сбалансированного дерева равен нулю.

(iii) Считаем, что реальная стоимость [в единицах потенциала] описанного в пункте (i) преобразования не α-сбалансированного поддерева T с m вершинами в 1/2-сбалансированное дерево равна m. Какой нужно выбрать константу c в зависимости от α, чтобы учётная стоимость такого преобразования T равнялась $O(1)$?

(iv) Покажите, что учётная стоимость удаления или вставки элемента в α-сбалансированное дерево с n вершинами равна $O(\log n)$.

Из предыдущей задачи вытекает, что для α-сбалансированных деревьев удаление или вставка выполняются эффективно в смысле учетной стоимости. Но можно построить другие более сложные древесные структуры данных, в которых эти операции теоретически выполняются быстрее. Например, в т. н. *фибоначчиевых* кучах **[Кормен 1, гл. 21] или [Кормен 2, гл 20]** удалить или вставить элемент удаётся с учётной стоимостью $O(1)$. Но, как и всегда, платой является более сложная организация программы. В ранее упомянутых АВЛ-деревьях[71] добавление требует $O(1)$, а удаление — $O(\log n)$ операций, но **в наихудшем случае**. В определённом смысле и фибоначчиевы кучи, и АВЛ-деревья входят в теормининимум по крайней мере для физтехов, претендующих на высокую оценку. Амортизационный анализ фибоначчиевых деревьев требует достаточно тонких аргументов, и его полезно посмотреть для закрепления изложенного материала.

[70] Контрольный вопрос: каким это таким стандартным способом?

Задание на 11-ю неделю: 15.04–21.04. [0.15]
Алгоритмы на графах
Литература: [Кормен 1], [Кормен 2, разд. 6], [ДПВ]

Поиск в глубину и поиск в ширину

Комментарий (особенно он предназначен для студентов, которые знают — или считают, что знают, — что такое поиск в ширину или в глубину). *Здесь мы изучаем качественные свойства указанных процедур, чтобы понять какую информацию о графе мы дополнительно получаем, когда запускаем один из этих естественных и с виду совершенно бесхитростных обходов вершин. В качестве простейшего контрольного вопроса можно спросить, почему обе процедуры* **линейные по входу**, *т. е. линейные по длине естественной кодировки графа: числу вершин и ребер. Вы должны быть аккуратны с ответом, например, уже потому, что обе процедуры предусматривают возвраты, так что каждая вершина или ребро может просматриваться не один раз.*

Если специально не оговорено, то рассматриваются т. н. простые неориентированные графы без петель и кратных рёбер.

Согласно Теореме 23.4 из книги **[Кормен 1]**, процедура *поиска в ширину*, начиная с данной вершины s, в графе[72] присваивает просматриваемым вершинам отметки, равные *кратчайшему пути (длина = число рёбер)* от s.

Задача 64. $(0.02 + 0.01)$ (i) Докажите или опровергните, что следующее условие дает критерий, когда остовное дерево $F \subseteq G$ является *деревом некоторого поиска в ширину* связного неориентированного графа G.

Остовное дерево $T \subseteq G$ является **деревом некоторого поиска в ширину** *связного неориентированного графа G, если и только если в нем можно выбрать одну из вершин s за корень так, чтобы T было деревом кратчайших путей из s в графе G. Иными словами, путь по дереву из s в произвольную вершину t содержит не больше рёбер, чем кратчайший путь между s и t в G.*

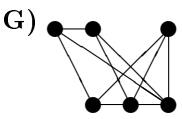
Если в настоящем виде критерий неверен, то модифицируйте его до корректного.

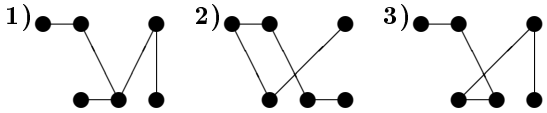
(ii) В соответствии с полученным в предыдущем пункте критерием установите, какие из нарисованных деревьев являются деревьями поиска в ширину.

Формат ответа. Пусть, скажем, критерий верен, тогда при положительном ответе нужно указать корень дерева кратчайших путей, а при отрицательном — для каждого возможного выбора корня нужно указать вершину, расстояние которой до корня в графе меньше, чем соответствующее расстояние по дереву.

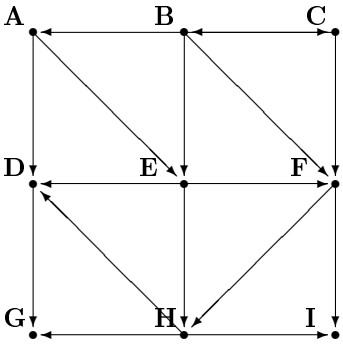
[71] Как обычно, рекомендую посмотреть книгу **[Ш]**. Она имеется в открытом доступе на сайте www.mccme.ru, и, хотя автор декларирует, что она написана для школьников, но объем, ясность и глубина изложенного в ней материала значительно превосходит стандарты изучения информатики в вузах.

[72] Или в ориентированном графе.





Задача 65. (0.02) Проведите поиск в глубину в графе на рисунке.



Используйте алфавитный порядок вершин. Укажите типы всех дуг графа и вычислите для каждой вершины значение функций $d(\cdot)$ и $f(\cdot)$.

Связность

Связностью или вершинной связностью $\kappa(G)$ неориентированного графа G называется наименьшее число вершин, удаление которых превращает граф в несвязный или тривиальный. **Рёберной связностью** $\lambda(G)$ графаG называется наименьшеее число рёбер удаление которых превращает граф в несвязный или тривиальный.

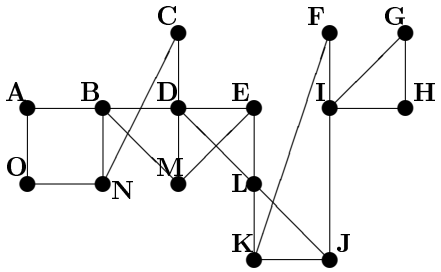
Максимальный по включению k-(реберно) связный подграф графа G называется его k- компонентой (соответственно, k-реберной компонентой). Обычно предполагается, что k-компонента имеет не менее $k+1$ вершин.

Задача 66. (0.01 балла) Покажите, что для любого G $\kappa(G) \leq \lambda(G) \leq \delta(G)$ ($\delta(G)$ — это минимальная степень вершин G).

Задача 67. (2×0.02 баллов) Постройте полиномиальный алгоритм или покажите NP-полноту проверки (i) k-связности и проверки (ii) k-реберной связности графа (k — двоичное число).

Точка раздела связного неориентированного графа G — это вершина, удаление которой делает граф несвязным. **Мост** — это ребро с аналогичным свойством. **Двусвязная компонента** связного графа содержит ≥ 3 вершин (или ≥ 2 ребер) и состоит из максимального набора ребер, в котором каждая пара ребер принадлежит общему простому (несамопересекающемуся) циклу.

Задача 68. (0.01) Для графа, изображённого на рисунке, укажите точки раздела, мосты и двусвязные компоненты.



k-связность графов

Ниже сформулированы утверждения, которые в принципе можно доказать, используя потоки в сетях, и речь идёт только о неориентированных графах.

Вершинной (соответственно, рёберной) связностью $\kappa(G)$ (соответственно, рёберной $\lambda(G)$) называется наименьшее число вершин (рёбер), удаление которых приводит к несвязному или тривиальному графу. .

В предыдущем задании мы установили неравенство $\kappa(G) \leq \lambda(G) \leq \delta(G)$ ($\delta(G)$ — максимальная степень вершин графа G).

Граф G называется вершинно n-связным или просто n-связным (соответственно, рёберно n-связным), если $\kappa(G) \geq n$ ($\lambda(G) \geq n$). Нетривиальный граф 1-связен, тогда и только тогда, когда он связен, и 2-связен, если и только если в нем более одного ребра и он не имеет точек сочленения. Например, полный граф K_2 не является 2-связным.

Попробуйте в качестве упражнения доказать, что граф двусвязен, тогда и только тогда, когда в нем любые две вершины принадлежат простому циклу.

Теоремы типа Менгера

Пусть u и v — две различные вершины связного графа G. Две простые цепи, соединяющие u и v, называются вершинно-непересекающимися, если у них нет общих вершин, отличных от u и v и рёберно-непересекающимися, если у них нет общих ребер. Множество S вершин, рёбер или вершин и ребер разделяет u и v, если u и v принадлежат различным различным компонентам графа $G \setminus S$.

Теорема 7 (Карл Менгер (1927)) *Наименьшее число вершин, разделяющих вершины u и v, равно наибольшему числу непересекающихся простых u–v цепей.*

Теорема 8 (Форд–Фалкерсон, Элайес–Файнштейн–Шеннон) *Для любых двух вершин графа наибольшее число рёберно-непересекающихся цепей, соединяющих их, равно наименьшему числу рёбер, разделяющих эти вершины.*

Теорема 9 (Хасслер Уитни) *Граф n-связен тогда и только тогда, когда любая пара его вершин соединена не менее, чем n вершинно-непересекающимися путями.*

Теорема 10 *Граф рёберно n-связен тогда и только тогда, когда любая пара его вершин соединена не менее, чем n рёберно-непересекающимися путями.*

Теорема 11 *Наибольшее число непересекающихся цепей, соединяющих два непустых непересекающихся вершин V_1 и V_2, равно наименьшему числу вершин, разделяющих V_1 и V_2.*

Назовём линией матрицы любую ее строку или столбец. Пусть M — $\{0,1\}$-матрица. Набор единичных элементов матрицы называется независимым, если никакая пара не лежит в общей линии.

Теорема 12 *В любой бинарной матрице наибольшее число независимых единичных элементов равно наименьшему числу линий, поекрывающих все единицы.*

Двусвязность. Сильная связность

Задача Д–25. (2×0.02) Дана выполнимая 2-КНФ φ, каждый дизъюнкт которой содержит ровно два различных литерала (литерал и его отрицание считаются различными). Будем говорить, что φ 1-*минимальна*, если к ней можно добавить один дизъюнкт, содержащий два различных литерала так, чтобы она стала невыполнимой.

(*i*) Докажите или опровергните, что следующее условие является критерием 1-минимальности.

Рассмотрим ориентированный граф G_φ, в котором литералы и их отрицания являются вершинами, а каждый дизъюнкт порождает пару ребер вида: $x \vee y \Rightarrow [e_1 = (\neg x, y), e_2 = (\neg y, x)]$.

φ является 1-минимальной тогда и только тогда, когда в G_φ есть путь P, соединяющий противоположные литеральные вершины, $x \rightsquigarrow y$, $x = \neg y$ и имеется ребро, ведущее из вершины y в вершину $z \notin P$.

Если в указанном виде критерий не верен, то дополните его до корректного.

(*ii*) Постройте для задачи проверки 1-минимальности как можно более быстрый полиномиальный алгоритм.

Подсказка. Полезно вспомнить, полиномиальные алгоритмы проверки выполнимости 2-КНФ.

Задача Д–26. (0.03) Постройте линейный по входу алгоритм, который, имея на входе граф G и некоторое его остовное дерево T, определяют, является ли T деревом поиска-в-ширину при старте с некоторой вершины G.

Задача Д–27. $(2\times0.01+0.02+0.01+2\times0.02)$ **Линейный алгоритм разбиения графа на двухсвязные компоненты**

(*i*) Покажите, что множества вершин, принадлежащие двум разным двусвязным компонентам, либо не пересекаются, либо имеют единственную общую вершину — точку раздела.

Построим по G новый граф G_b, в котором имеются вершины двух типов: v_a, отвечающие точкам раздела G, и v_b, отвечающие двусвязным компонентам G. Ребра G_b соединяют каждую вершину v_b со всеми вершинами v_a, попадающими в двусвязную компоненту, отвечающую v_b.

(*ii*) Покажите, что G_b — дерево, и постройте соответствующее дерево для G из задачи **№ 68**.

Оказывается, что точки раздела можно находить по дереву поиска в глубину. Затем, опять используя поиск в глубину, можно определить все двусвязные компоненты, т. е. двусвязные компоненты можно находить за линейное время. Мы ограничимся только алгоритмом выделения точек раздела графа.

(*iii*) Докажите, что корень дерева поиска в глубину является точкой раздела тогда и только тогда, когда у него больше одного потомка.

(*iv*) Постройте контрпример к следующему утверждению из книги **[Кормен 1, задача № 23-2 (б)]**: отличная от корня вершина v дерева поиска в глубину является точкой раздела, если и только если в дереве поиска в глубину не существует обратного ребра от потомка v (включая саму v) до собственного предка v (т. е. отличного от самой v).

(*v*) **[Кормен 1, упр. 23-2(в)]**.

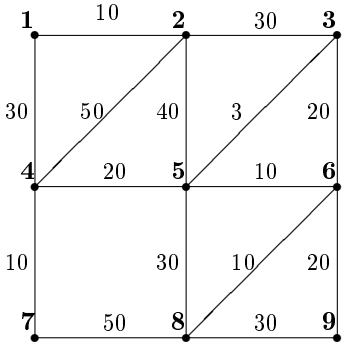
Определим функцию $low(v) = \min[d(v), d(w)$, *если для некоторого потомка и вершины v в G есть обратное ребро* $(u, w)]$.

Покажите, как вычислить $low(\cdot)$ за время $O(|E|)$ [например, модифицируя поиск в глубину].

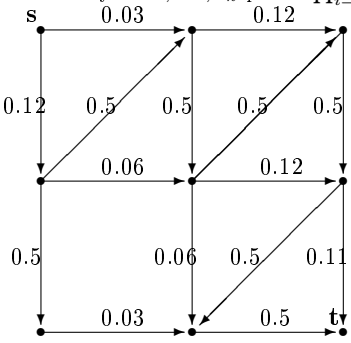
(*vi*) Покажите, как в линейное время вычислить все двусвязные компоненты графа[73]

Кратчайшие пути и связывающие сети

Задача 69. (2×0.01) Найдите минимальное остовное дерево взвешенного графа G на рисунке с помощью алгоритмов Прима и Краскала. Изобразите графы, полученные на трех последовательных итерациях алгоритмов.



Задача 70. $(2 \times 0.02 + 0.03)$ Коммуникационная сеть является ориентированным графом, причём каждому ребру (каналу связи) (u, v) приписано число $r(u, v)$— "надежность соединения", где $0 \le r(u, v) \le 1$, так что $1 - r(u, v)$ можно рассматривать как вероятность разрыва соединения при передаче. Считаем, что "вероятности" $r(u, v)$ независимые, и таким образом, надежность передачи сообщения по пути $v_1, \ldots, v_k$ равна $\prod_{i=1}^{k-1} r(v_i, v_{i+1})$.



(*i*) Постройте эффективный алгоритм нахождения наиболее надёжного пути в сети между вершинами s и t и укажите класс сетей, в которых алгоритм будет эффективным.

(*ii*) Проведите вычисления по вашему алгоритму для сети, изображённой на рисунке.

(*iii*) Постройте наиболее надёжную сеть, позволяющую передавать сообщения из вершины s в любую другую вершину графа, содержащую **минимальное** число дуг. Под *надёжностью* сети понимается произведение надёжностей всех входящих в нее дуг.

Вершины ориентированного **грид-графа** расположены в целых точках плоскости: $V(G) = \{(i, j), i = 0, \ldots, m, j = 0, \ldots, n\}$, а дуги соединяют соседние точки: $E(G) = \{[(i, j) \to (i + 1, j)], i =$

[73] **Подсказка.** Сначала, используя решение предыдущих задач, покажите, как с помощью поиска в глубину найти все точки раздела за линейное время. Это и свойства функции $low(\cdot)$ позволяют находить двусвязные компоненты при поиске в глубину, используя дополнительный стек. Можно также находить двусвязные компоненты другим способом (см. **[Кормен I, упр. 23-2(д)–(з)]**).

$0,\dots,m-1$, $j=0,\dots,n$ или $[(i,j)\to(i,j+1)]$, $i=0,\dots,m$, $j=0,\dots,n-1\}$, причём дугам G приписаны целочисленные веса. Рассмотрим задачу поиска экстремального (самого "тяжелого" или самого "лёгкого") пути между вершинами $(0,0)$ и (m,n) (по определению, вес пути равен сумме весов входящих в него рёбер). В таком виде — это типичная задача так называемого "динамического программирования", описанная во многих источниках. Для неё легко придумать оптимальный $O(mn)$-алгоритм

Задача 71. (2×0.01) (*i*) Постройте $O(mn)$-алгоритм поиска экстремального пути.
(*ii*) Покажите, что ваш алгоритм оптимальный по сложности, поскольку любой алгоритм, решающий задачу, обязан прочитать вход (таблицу весов, размер которой равен mn). Иначе говоря, нужно показать, что ответ существенно зависит от каждого входного параметра.

Задача Д–28. (0.3) (Это задача № Д13 из методички.) Пусть теперь веса всех горизонтальных дуг $[(i,j)\to(i+1,j)]$ зависят только от j, а веса всех вертикальных дуг $[(i,j)\to(i,j+1)]$ зависят только от i. Таким образом, веса полностью заданы, если указаны n "горизонтальных" весов u_j и m "вертикальных" весов v_i, и длина входа равна в этом случае $O(m+n)$.

Постройте оптимальный линейный $O(m+n)$-алгоритм вычисления экстремальных путей для этого класса гридграфов.

Структуры UNION-FIND
Амортизационный анализ

Для эффективной реализации алгоритма Краскала нужно научиться эффективно обрабатывать непересекающиеся множества. Для этого был придуман специальный алгоритм, который при опросе на одной референтном профессиональном сайте с большим отрывом был признан наиболее выдающимся (это, видимо, достаточный повод, чтобы с ним ознакомиться).

Мы хотим реализовать следующие операции над множествами.

- $MAKESET(x)$ — создать множество с единственным элементом x;
- $UNION(x,y)$ — заменить множества с именами x и y их объединением
- $FIND(x)$ — возвратить имя множества, содержащего элемент x;
- $LINK(x,y)$ — (x и y — корни) перевесить указатель корня x на корень y; в этих обозначениях $UNION(x,y) = LINK(FIND(x),(FIND(y))$.

Можно действовать совсем бесхитростно и считать, что множества — это, например, связные списки, имеющие специальный указатель на *имя множества*. Тогда, грубо говоря, поиск, объединение и пр. пропорционален (суммарному) размеру участвующих в операции множеств. Понятно, что это может быть весьма медленным. Попробуем действовать чуть похитрее (и замечу, вполне практически разумно. Для этого введём специальную характеристику множества: его размер и будем при объединении присваивать *ме́ньшему* множеству имя *бо́льшего*.

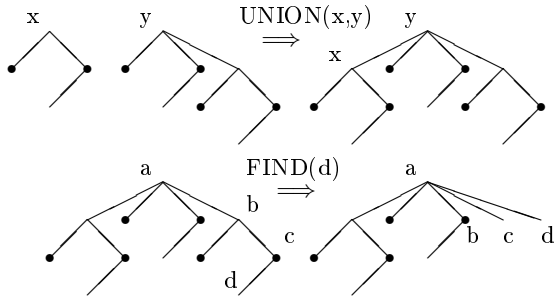
Внимание! При решении следующей задачи вы должны уточнить детали и описать конкретную структуру данных и алгоритм, реализующий это предложение! Предполагается, что все множества являются подмножествами $\{1,\dots,n\}$ **и над ними проводятся только операции** UNION-FIND, **причём объединяются только непересекающиеся множества.**

Задача 72. (0.03) Постройте алгоритм, который выполняет ($k\le n-1$) операций UNION и m операций FIND и имеет трудоёмкость $O(\max(m,n\log n))$.

Если $m=O(n\log n)$, то построенный в предыдущей задаче алгоритм является оптимальным по трудоёмкости, но, если $m=O(n)$, то процедуру можно ускорить и предложить алгоритм, выполняющую $O(n)$ операций UNION-FIND за почти линейное время. Для этого вместо связного списка нужно представлять множества лесом корневых (ориентированными) деревьев, ребра которых (указатели в вершинах) направлены к предку. Корень отождествляется с именем множества и его указатель замкнут на себя. Будем считать, что трудоемкость операции FIND равна числу просмотренных в дереве элементов, т. е. не превышает высоты дерева, а сложность UNION равна сложности двух операций FIND. Если теперь использовать изложенный выше прием и при объединении перемещать указатель корня в дереве ме́ньшей высоты на корень дерева бо́льшей высоты, то можно получить следующий результат (опять в силе остаётся требование формального описания структуры данных, алгоритма и оценки его трудоёмкости)

Задача Д–29. (0.03) Постройте алгоритм, который выполняет $O(n)$ операций UNION и FIND и имеет трудоёмкость $O(n\log n)$.

Важно понять, что предыдущий результат был получен потому, что нам удалось поддерживать структуру данных, в которой представляющие множества деревья имеют маленькую высоту. Сам приём объединения деревьев по высоте, называется алгоритмом ОБЪЕДИНЕНИЯ ПО РАНГУ, и, как мы видим, он и практичен, и достаточно эффективен. Но оказывается, что если дополнить его ещё одной процедурой, называемой СЖАТИЕ ПУТЕЙ, то мы получим для той же задачи ещё более быстрый, почти линейный алгоритм, который и возглавляет список "Алгоритмов из КНИГИ". На самом деле, такая высокая оценка, видимо, даётся не только за элегантность процедуры, но и за очень интересный метод оценки трудоёмкости. Неформально, первая процедура "привешивает" более низкие деревья с меньшей высотой (=рангом) к более высоким, а вторая,— при поиске любого элемента дерева сразу перевешивает его указатель на корень, как показано на рисунке.



Формальное описание следующее.

```
procedure MAKESET(x)
p(x):=x
rank(x):=0
end

function FIND(x)
if x не равно p(x) then
p(x):=FIND(p(x))
Выполняем сжатие путей, т. е. направляем на
корень указатели всех элементов в пути от корня до x
return(p(x))
end

function LINK(x,y)
if rank(x))>rank(y) then  обменять x и y местами
if rank(x)=rank(y) then rank(y):=rank(y)+1
p(x):=y
return(y)
end
```

```
procedure UNION(x,y)
LINK(FIND(x),FIND(y))
end
```

Заметим, что каждая операция FIND или UNION для множеств из n элементов может выполняться за время $O(\log n)$, Но ниже, **используя, как в своё время было обещано, амортизационный анализ**, мы покажем, что выполнение m операций FIND или UNION требует не более $O((m+n)\log^* n)$ операций. А на самом деле, и ещё меньше: вместо $\log^* n$ можно поставить обратную к функции Аккермана, рост которой совершенно ничтожен даже по сравнению с $\log^* n$.

Изучим свойства ранга.

- по определению, если $v \neq p(v)$, то $rank(p(v)) > rank(v)$;
- по определению, если $p(v)$ изменяется, то $rank(p(v))$ увеличивается.

Задача 73. (3×0.01) (i) Докажите, что число элементов ранга k не превышает $\frac{n}{2^k}$.
(ii) Докажите. что число элементов ранга $\geq k$ не превышает $\frac{n}{2^{k-1}}$.
(iii) Докажите, что ранг произвольного элемента не превышает $\log n$.

Теперь используем амортизационный анализ, и будем оценивать трудоёмкость не одной, а сразу m операций FIND или UNION. Поскольку, как мы уже говорили, последняя выражается через LINK и две операции FIND, то достаточно оценить сложность $2m$ операций FIND (LINK, требует $O(1)$ операций).

Но сначала нужно понять, в чем основная трудность. Дело в том. что FIND *— рекурсивная процедура. Кроме того, и это выглядит не только угрожающим, но и совершенно безнадёжным, нам нужно оценить трудоёмкость процедуры, в которой текущий лес, представляющий систему непересекающихся множеств, изменяется с каждой операцией (и тем самым изменяется трудоёмкость конкретных поисков и объединений).*

Ниже приведён набросок оценки. Его нужно продумать, ибо он нетривиальный и содержит тонкие места. Грубо говоря, мы применим хитрый бухгалтерский трюк: мы разделим алгоритм на этапы, через которые должна проходить любая операция, и будем оценивать трудоёмкость каждого этапа сразу для всех m операций. Поскольку речь идёт о том, что нужно оценить сложность последовательности операций FIND, т. е. длину путей в соответствующих деревьях, то этапы заключаются в разделении всего возможного диапазона высот деревьев на части и подсчёт операций в каждой части.

Формальная конструкция. Ранг элемента не меняется, как только он перестаёт быть корневым. Разделим все некорневые элементы на группы по величине их ранга r, причём отнесём в группу i все элементы, для которых выполнено равенство $\log^* r = i$, т. е. в i-ю группу попадут элементы с рангами из полуинтервала $(2^{i-1}, 2^{2^{i-1}}]$. Ниже будем использовать сокращение $k = 2^{i-1}$.

Задача Д–30. (2×0.01) (i) Покажите. что число различных групп не превышает $\log^* n$.
(ii) Число элементов в i-й группе не превышает $\frac{n}{F(i)}$.

Теперь используем следующие обозначения, пусть $\Sigma = \{\sigma_i,\ i = 1\ldots, m\}$, где σ_i — это i-й оператор FIND последовательности Σ длины m. σ_i индуцирует путь L_i в соответствующем текущем дереве от найденного элемента до корня. Формула ниже обслуживает сразу великое множество допустимых последовательностей FIND, и, что выглядит совершенно неправдоподобным,— её удаётся проанализировать. Итак, подсчитаем для любой последовательности Σ следующую величину (знак "$\rightarrow$"над суммой, говорит о том, что величины зависят от порядка операторов, которые изменяют деревья в процессе поиска).

$$\vec{\sum}_{\sigma_i\in\Sigma}\ [\text{ число таких } (u\in L_i)\wedge(u,p(u)) \text{ из одной группы }]+$$
$$\vec{\sum}_{\sigma_i\in\Sigma}\ [\text{ число таких } (u\in L_i)\wedge(u,p(u)) \text{ из разных групп или u — корень}].$$

Пояснение. Понимать это выражение можно следующим образом. Давайте мысленно покрасим каждый оператор FIND и все рекурсивные вызовы (путь по дереву к корню), которые он порождает, своим цветом, так что можно говорить о траектории. Конечно, достаточно, скажем, изменить первый оператор в последовательности и вся картинка (и траектории), возможно, изменится. Тем не менее, каждая траектория пересекает границы групп (число таких рёбер-пойнтеров подсчитывается во второй сумме) и, кроме того, поскольку, по построению, при сжатии путей ранги родителей монотонно возрастают по крайней мере на единицу, то любой элемент из i-й группы может быть подвергнут процедуре сжатия путей не более $F(i)-F(i-1)$ раз, прежде чем он получит родителя из следующей группы (и тогда трудоёмкость просмотра элементов будет учитываться во второй сумме).

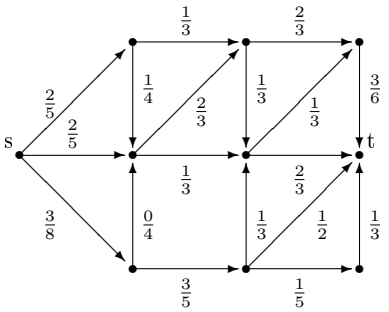
Вторая сумма оценивается как $O(m\log^* n)$ **просто потому. что число групп (уровней) по построению $O(\log^* n)$, а мы подсчитываем,"события", когда ребра-пойнтеры пересекают границы групп**. А первую сумму можно оценить следующим образом. Число операций, которым может подвергаться отдельный элемент внутри группы по порядку равен его рангу

$$\sum_{\text{по всем группам}} [\text{число элементов в группе}]\times$$
$$[\text{максимальный ранг элементов группы}] \leq \sum_{k=1}^{\log^* n}\frac{n}{F(k)}F(k) \leq n\log^* n,$$

откуда получаем требуемую трудоёмкость $O((m+n)\log^* n)$.

Задание на 12-ю неделю: 22.04–28.04. [0.08]
Потоки и разрезы
Линейное программирование
Литература: [Кормен 1], [Кормен 2, гл 26], [ДПВ]

Задача 74. $(3\times 0.01 + 0.02 + 0.01)$ На рисунке изображён потоковый граф (метка $\frac{f}{u}$ на ребре означает поток и пропускную способность, соответственно).



(i) Чему равен поток f?
(ii) Изобразите остаточный граф, соответствующий потоку f.
(iii) Максимален ли поток f?

В следующих двух пунктах нужно по шагам применить метод[74] Форда-Фалкерсона, доведя его до алгоритма. **При отсутствии**

[74] Отметим, что выражение "метод" употребляется не случайно (некоторые этапы описаны неявно или подразумеваются). **Вы должны самостоятельно придумать, как дополнить процедуру до алгоритма.**

алгоритма (например, если увеличивающие пути находятся методом "внимательного рассмотрения" потокового графа или если разрез просто отгадывается) задача не оценивается. *Это требование будет тем более актуально при написании тестов.*

Метод остаточных графов из книг **[Кормен 1]** или **[Кормен 2]** аналогичен оригинальному методу пометок Форда- Фалкерсона, изложенному в их книге

(*iv*) С помощью алгоритма Форда-Фалкерсона по шагам найдите максимальный поток. На каждом шаге должен быть построен остаточный граф и указан увеличивающий путь.

(*v*) Укажите модификацию алгоритма Форда-Фалкерсона для нахождения минимального разреза. По шагам постройте минимальный разрез между s и t. Найдите его пропускную способность.

Задача 75. (0.02 + 0.01) В больнице каждому из 169 пациентов нужно перелить по *одной дозе* крови. В наличии имеется 170 доз. Распределение по группам таково.

Группа	I	II	III	IV
В наличии	45	32	38	55
Запрос	42	39	38	50

При этом пациенты, имеющие кровь группы I, могут получать только кровь группы I. Пациенты, имеющие кровь группы II (группы III), могут получать только кровь групп I и II (групп I и III, соответственно). Наконец, пациенты с IV группой могут получать кровь любой группы.

(*i*) Распределите дозы, чтобы обслужить максимальное число пациентов с помощью *решения подходящей задачи о максимальном потоке*. Решение нужно аккуратно оформить: должна быть нарисована потоковая сеть и показаны все шаги алгоритма ФФ, начиная с нулевого потока, т. е. должны быть построены остаточные графы и показаны увеличивающие пути.

(*ii*) Если всех пациентов обслужить нельзя, то приведите *простое* объяснение этому, *доступное администрации* больницы.

Задача 76. (0.01) Покажите на примере конкретной сети, что алгоритм Форда-Фалкерсона не является полиномиальным.

Рассмотрим следующую задачу Сеть. Дан ориентированный граф $G = (V, E)$, дугам которого приписаны неотрицательные числа $l_i \le u_i, i \in E$. Нужно проверить, можно ли приписать ребрам числа $F = \{f_i, i \in E\}$, чтобы в любой вершине v была нулевая дивергенция $div\, F = \sum_{\text{по входящим в v ребрам}} f_i - \sum_{\text{по выходящим из v ребрам}} f_j = 0$ и выполнялись неравенства $l_i \le f_i \le u_i, i \in E$.

Задача 77. (0.01 + 0.02) Покажите, как можно решить задачу Сеть с помощью решения подходящей задачи о максимальном потоке в сети и наоборот.

Задача 78. (0.01) Рассмотрим следующую задачу. В потоковой сети нет ограничений пропускной способности на дугах, но есть ограничения пропускной способности вершин. Формально, для каждой вершины v, отличной от истока и стока, задано целое неотрицательное число $c(v)$, и для потока в сети должно выполняться

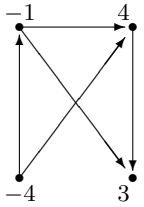
$$\sum_u f(u,v) = \sum_u f(v,u) \le c(v).$$

Опишите алгоритм нахождения максимального потока в такой сети.

Задача Д–31. (0.02) Задан двудольный неориентированный граф, в котором обе доли имеют n вершин, а степени (количество инцидентных рёбер) всех вершин равны d, **т. е. однородный двудольный граф степени** d. Приведите полиномиальный алгоритм, который раскрасит рёбра в d цветов так, чтобы из каждой вершины исходили рёбра разных цветов. Оцените сложность предложенного алгоритма.

Задача Д–32. (0.02) На вход задачи подаётся ориентированный граф $G = \langle V, E\rangle$ без контуров (ориентированных циклов). Необходимо покрыть его наименьшим числом простых путей, т. е. найти наименьшее количество не пересекающихся по вершинам простых путей, чтобы каждая вершина принадлежала одному из них. Допускаются пути нулевой длины (состоящие из одной вершины). Предложите полиномиальный алгоритм.

Задача 79. (0.02 + 0.01 + 0.02) Последовательность выполнения проектов задана ациклическим орграфом $G = (V, E)$ (если в орграфе есть ребро (u, v), то проект v не может начаться, пока не будет выполнен проект u). Выполнение проекта v приносит прибыль $p(v)$ (она может быть и отрицательна). Требуется выбрать подмножество проектов, приносящих максимальную суммарную прибыль, т. е. найти такое подмножество проектов $M \subseteq V$, что $M = argmax_{S\subseteq V}\{p(S) \stackrel{def}{=} \sum_{v\in S} p(v)\}$.



Оказывается, что эту задачу можно свести к задаче о минимальном разрезе. **Конструкция.** *Дополняем граф G источником s и стоком t, и задаем* **бесконечные** *пропускные способности на ребрах G. Далее, для всех вершин $v \in V$, если $p(v) < 0$, то задаем ребро (s, v) с пропускной способностью $-p(v)$, а если $p(v) > 0$, то задаем ребро (v, t) с пропускной способностью $p(v)$.*

(*i*) С помощью алгоритма Форда-Фалкерсона найдите максимальный поток в полученной сети. Начальный поток нулевой. Приведите подробное описание: на каждом шаге процедуры на вспомогательных чертежах изобразите остаточные графы и укажите увеличивающие пути.

(*ii*) Затем, используя алгоритм Форда-Фалкерсона, найдите минимальный разрез.

(*iii*) Обоснуйте конструкцию в общем случае.

Задание на 13-ю неделю: 29.04–5.05. [0.08]
Потоки и разрезы
Линейное программирование
Литература: [Кормен 1], [Кормен 2, гл 26], [ДПВ]

Задача 80. Проведите через точки $(1, 3), (2, 5), (3, 7), (5, 11), (7, 14), (8, 15), (10, 19)$ *наименее уклоняющуюся прямую*. Иначе говоря, решите следующую задачу: $\max_{(a,b,c)\in\mathbb{R}^3} \min_{1\le i\le 7} \mid ax_i + by_i - c \mid$, здесь (x_i, y_i) — координаты точек.

Комментарий. На физлабах нас учат проводить наилучшие прямые или плоскости, используя метод наименьших квадратов. При этом отклонение измеряется квадратом расстояния. Но можно, а в некоторых задачах и предпочтительнее, вычислять абсолютное отклонение, как требуется в этой задаче.

Задача 81. Многогранник $P_\varepsilon \subset \mathbb{R}^3$ задан неравенствами: $0 \le x_1 \le 1, \varepsilon x_1 \le x_2 \le 1 - \varepsilon x_1, \varepsilon x_2 \le x_3 \le 1 - \varepsilon x_2$, здесь $\varepsilon \in (0, \frac{1}{2})$. Геометрически P_ε — это деформированный куб. Покажите, что в кубе P_ε есть путь по ребрам, стартующий из начала координат и проходящий по всем вершинам, в котором величина координаты x_3 монотонно возрастает.

Комментарий. Этот пример и его обобщения часто используют для того, чтобы показать, что многие алгоритмы линейного программирования типа симплекс-метода могут быть экспоненциальными.

Задача 82. Пусть A — матрица порядка $m \times n$. Покажите, что имеет место альтернатива: из пары следующих систем разрешима только одна.

$$\begin{aligned} A^T p &\le 0, p \in \mathbb{R}^m \\ Ay = 0, \quad y \ge 0, &\quad y \in \mathbb{R}^n, y \ne 0. \end{aligned} \tag{6}$$

Комментарий. Пусть $H = \{y \in \mathbb{R}^n \mid Ay = 0\}$ — векторное подпространство и $H^\perp$ — его ортогональное дополнение. Назовём вектор из $\mathbb{R}^n$ положительным, если все его компоненты положительны. Тогда альтернатива утверждает, что однородная система (6) несовместна тогда и только тогда, когда $H^\perp$ пересекает внутренность неотрицательного ортанта $\mathbb{R}^m_+$. Иными словами, доказательством (сертификатом) того, что однородная система (6) не имеет нетривиальных неотрицательных решений является существование произвольного положительного вектора $a \in H^\perp$, $a > 0$. Посредством достаточно стандартных манипуляций проверка совместности произвольной системы линейных неравенств или *задача линейного программирования* полиномиально сводится к проверке совместности (6), и поэтому любую полиномиальную процедуру нахождения сертификата "a" можно в принципе конвертировать в полиномиальный алгоритм линейного программирования. Отсюда и следующая задача.

Задача 83. Пусть e_i, $i = 1 \ldots, n$ — орты $\mathbb{R}^n$, и $e_0 = e_1 + e_2 + \cdots + e_n$. Докажите или опровергните, что если пересечение $H \cap \mathbb{R}^n_+ = \{0\}$ тривиально, то проекция на подпространство $H^\perp$ хотя бы одного из векторов $e_0, e_1, \ldots, e_n$ положительна. Отдельно проверьте утверждение для $n = 2, 3$.

Задача 84. Докажите, что для любого натурального k в $\mathbb{R}^4$ существует выпуклый многогранник, имеющий k вершин (крайних точек), причём любая пара из них соединена ребром.

Задача Д–33. На плоскости заданы n белых и m черных точек. Постройте $O(n+m)$-алгоритм, который находит, если это возможно, прямую, отделяющую каждую белую точку от каждой чёрной.

Задание на 14-ю неделю: 6.05–12.05. [0.09]
Методы решения переборных задач
Ещё о жадных алгоритмах
Литература: [К-Ф], [Кормен 2, §§15, 16] [Кормен 1], [ДПВ]

Идущая ниже задача посвящена дискретной оптимизационной *задаче о рюкзаке (ЗР).* **Она интересна тем, что в ней используется около половины известных алгоритмических стратегий решения т. н. "переборных" задач: "жадная", "релаксация целочисленных ограничений", "гарантированная оценка", "динамическое программирование", "шкалирование", "ветви-и-границы". Подробное решение можно прочитать в книге [К-Ф]**

Полезно хотя бы ознакомиться с задачей, поскольку в той или иной форме она может включена в контрольные.

Задача 85. $(6 \times 0.01 + 0.03)$.

Пусть заданы положительные целые векторы $a, c \in \mathbb{Z}^n$ и натуральное число b. *Предполагается, что b не меньше минимальной компоненты вектора a.* Требуется найти:

$$\begin{aligned} &c_1x_1 + \cdots + c_nx_n = cx \to \max \\ &a_1x_1 + \cdots + a_nx_n = ax \le b \\ &x = (x_1, \ldots, x_n) \in \{0,1\}^n \end{aligned}$$

Неформально говоря, мы хотим заполнить рюкзак ограниченного объема наиболее ценными вещами. Если последнее ограничение заменить на $0 \le x_i \le 1$, $i = 1, \ldots, n$, то получится *непрерывная релаксация дискретной задачи о рюкзаке (НЗР).*

(i) Докажите, что НРЗ эффективно решается с помощью "жадного" алгоритма. Алгоритм нужно придумать, доказать его корректность (обычно вместо доказательства приводится набор лозунгов) и оценить трудоемкость.)

(ii) Покажите, что простая модификация "жадного" алгоритма из предыдущей задачи позволяет найти $\frac{1}{2}$-*приближенное* решение ЗР[75], т .е. найти такой $\{0,1\}$-вектор $\tilde{x}$, что

- $a\tilde{x} \le b$;
- $\frac{c\tilde{x}}{cx_{opt}} \ge \frac{1}{2}$,

здесь x_{opt}– оптимальное решение ЗР.

Постройте псевдополиномиальные алгоритмы[76] для решения ЗР

(iii) с трудоемкостью $O(nb)$;

(iv) с трудоемкостью $O(nf_{opt})$, здесь f_{opt} — оптимальное решение ЗР.

(v) Для заданного $\varepsilon > 0$ выберем максимальное натуральное число t из условия $n2^t \le \varepsilon f_{opt}$. Отбросим у коэффициентов целевой функции ЗР все младшие t разрядов[77]: $c_i \to \tilde{c}_i = \lfloor \frac{c_i}{2^t} \rfloor 2^t$ Пусть $\tilde{f}$– оптимальное решение ЗР с измененной целевой функцией $\tilde{c}_1x_1 + \ldots \tilde{c}_nx_n$. Покажите, что $|\tilde{f} - f_{opt}| \le \varepsilon f_{opt}$.

(vi) Используя пункты (ii), (iv), (v), постройте алгоритм, находящий ε-*приближенное решение* ЗР, т. е. такой вектор $\tilde{x} \in \{0,1\}^n$, что

- $a\tilde{x} \le b$ ($\tilde{x}$– допустимое решение ЗР);
- $f_{opt} - c\tilde{x} \le \varepsilon f_{opt}$.

[75] Подчеркиваем, "жадный" алгоритм **не дает** $\frac{1}{2}$-приближенное решение ЗР. Полезно рассмотреть пару примеров, и сразу же станет понятно, как исправить процедуру.

[76] Так называются процедуры, трудоемкость который зависит от от унарной, а не двоичной длины записи некоторых используемых в описании числовых параметров. В идущих ниже пунктах такими параметрами являются, соответственно, b и f_{opt}.

[77] Это преобразование называется *шкалированием (scaling).*

Трудоемкость алгоритма должна быть $O(\frac{n^2}{\varepsilon})$ операций над $O(\max\{b, \frac{n}{\varepsilon}\})$ числами, **используя память** $O(\frac{n}{\varepsilon})$**.**

Процедура с такими характеристиками (сложность ε-приближенного алгоритма оценивается полиномом от длины входа и $\frac{1}{\varepsilon}$) называется **полностью полиномиальной приближенной схемой (ППТС) — fully polynomial approximation scheme.**

(*vii*) Используя построенный алгоритм, найдите оптимальное решение следующей задачи[78] о рюкзаке:

$$\varepsilon = 0.25$$
$$58x_1 + 70x_2 + 59x_3 + 8x_4 + 60x_5 + 240x_6 + 250x_7 + 249x_8 \quad \to \quad \max$$
$$33x_1 + 5x_2 + 34x_3 + 5x_4 + 31x_5 + 185x_6 + 190x_7 + 191x_8 \quad \leq \quad 256$$
$$x_i \in \{0, 1\}$$

Еще о "жадных" алгоритмах

С "жадными" алгоритмами вы уже сталкивались. Например, это алгоритмы Прима или Краскала. Ниже мы приведём ещё один пример, когда они работают, а именно, рассмотрим задачу поиска достаточно большого множества независимых вершин в планарном графе. Напомним известное утверждение, которое уже использовалось нами ранее.

Факт. *Число рёбер e (а также число плоских граней f при любом вложении) произвольного планарного графа равно $O(v)$. (Утверждение верно для графа любого фиксированного рода).*

Доказательство использует формулу Эйлера и простое неравенство $3f \leq 2e$, поскольку число ребер в каждой грани при вложении не превышает трёх и каждое ребро считается два раза.

Рассмотрим задачу нахождения независимого множества в графе, т. е. задачу нахождения множества, не связанных между собой (независимых) вершин. Поскольку вместо исходного графа можно перейти к дополнительному, в котором независимые множества переходят в клики, то язык НЕЗАВИСИМОЕ МНОЖЕСТВО $\{(G(V, E), k)$: в графе G есть независимое множество мощности $k\}$ является NP-полным.

Задача Д–34. (0.02) Покажите, что в планарных графах можно эффективно находить независимое множество мощности $const|V|$ ($|V|$ — число вершин графа, а константу можно считать равной $\frac{1}{25}$).

Подсказка: из Факта следует, что в планарных графах мало рёбер, а потому есть много несвязанных вершин, поэтому можно попробовать действовать совершенно бесхитростно: упорядочить вершины по возрастанию степеней и, последовательно двигаясь по этому списку, набирать множество несвязанных вершин, пока это возможно. Теперь осталось оценить размер полученного независимого множества.

В предыдущей задаче мы искали тупиковое максимальное независимое множество с помощью жадного алгоритма. А можно ли эффективно искать не *максимальное по включению (нерасширяемое)* независимое множество, а *максимальное по размеру, наибольшее*[79] независимое множество вершин какого-нибудь планарного графа посредством аналогичной процедуры? С нашим небольшим опытом уже должно быть понятно (особенно, если учесть, что уже для кубических планарных графов задача поиска самого большого по размеру независимого множества NP-трудна), что таким простым методом действовать нельзя. Тем не менее, остаётся надежда, что жадный алгоритм позволит найти нечто "близкое" к оптимуму. Заметим, что в случае планарных графов жадные алгоритмы достаточно эффективны в смысле точности приближенного решения. В самом деле, в предыдущей задаче мы показали, что жадный алгоритм находит независимое множество мощности $O(n)$ и ошибается разве что в константу раз. Как мы увидим из следующей задачи, такая ситуация **совершенно исключительная**. Для общих графов алгоритм может ошибаться в $O(n)$ раз, причём[80], если $\mathcal{P} \neq \mathcal{NP}$, то *вообще не может быть никакого полиномиального алгоритма, имеющего точность (мультипликативную погрешность) ниже $O(n)$.*

Вдумайтесь, пожалуйста в последнее утверждение. Речь идёт о том, что никакая даже самая продвинутая полиномиальная процедура, даже использующая случайный выбор, не может иметь (если $\mathcal{P} \neq \mathcal{NP}$) гарантированную ошибку, меньшую, чем самая тупая. Немного утрируя: можно затратить 50 лет на написание изощрённого алгоритма, который, тем не менее, при обработке графа G с одним гуглом вершин найдёт в нем только две независимые вершины, а на самом деле, в G их может быть, скажем, гугл/10.

Задача Д–35. (2×0.02) **[Иногда "жадные" алгоритмы крайне плохи и это стандартная ситуация!].**

(*i*) Рассмотрим "жадный" алгоритм для задачи нахождения наибольшего *независимого (внутренне устойчивого)* множества в неориентированном графе G. На каждом шаге обрабатывается пара (I_i, G_i), $i = 1, 2, \ldots$ и $I_1 = \emptyset$, $G_1 = G$. На i-м шаге в в графе G_i $i = 1, 2, \ldots$ выбирается произвольная вершина v_i минимальной степени и добавляется в $I_{i+1} \leftarrow I_i \cup \{v_i\}$, а G_{i+1} образуется из G_i путем удаления v_i и всех смежных с ней вершин.

Постройте граф, имеющий $O(n)$ вершин и независимое множество размера $\Omega(n)$, в котором "жадный" алгоритм находит независимое множество размера $O(1)$.

(*ii*) Рассмотрим "жадный" алгоритм для нахождения хроматического числа неориентированного графа G. Сначала множество вершин упорядочивается: $\{v_1, v_2, \ldots, v_n\}$ некоторым способом. На i-м шаге $i = 1, \ldots, n$ строится окраска *индуцированного подграфа* G_i с множеством вершин $V_i = \{v_1, \ldots, v_i\}$ (и "старыми" ребрами) следующим способом. Выбирается следующая по порядку вершина v_i, и если соседи вершины v_i в G_i окрашены всеми цветами, использованными для G_{i-1}, то v_i красится в новый цвет, иначе, — для нее используется один из старых цветов так, чтобы получить корректную окраску G_i.

Постройте пример графа, имеющего $O(n)$ вершин и хроматическое число $O(1)$, в котором "жадный" алгоритм при некотором упорядочении вершин находит окраску в $\Omega(n)$ цветов. *При этом вершина v_i должна при каждом выборе иметь минимальную степень в G_i.*

Оказывается, что "жадный" алгоритм, описанный в пункте (*ii*) корректно красит планарные графы в 6 цветов. Вообще-то планарные графы можно красить и в четыре цвета, и, собственно, доказательство одновременно является и квадратичным $O(n^2)$ алгоритмом окраски. Сам алгоритм был предложен почти 40 лет назад и основан на большом объёме (незамкнутых) компьютерных вычислений и считается корректным. Существует красивый $O(n \log n)$-алгоритм 5-окраски планарного графа, к тому же допускающий параллельную реализацию.

Ещё один знаменитый жадный алгоритм связан с задачами сжатия информации — это т. н. код Хаффмена. Прочитать о нем можно в (**[Кормен 1, §17.3]** или **[Кормен 2, §16.3]**).

[78] Приведите ВСЕ шаги процедуры. Решение, использующее другую ППТС, оценивается в **0.02 балла**, а решение любым другим способом в **0.01 балла**. Заметим, что все выкладки могут быть произведены практически в уме.

[79] По-английски последнее называется maximum в отличии от тупикового — maximal.

[80] Идущее ниже утверждение и его обобщения, т. н. PCP-теорема, полученные в начале 90-х годов прошлого века считаются самым крупным достижением теории вычислений прошлого века.