П Р О Г Р А М М А

1. Примеры алгоритмов: проверка простоты, факторизация чисел; одновременное вычисление максимального и минимального элементов в массиве; быстрое умножение чисел и матриц (алгоритмы Карацубы и Штрассена); аддитивные цепочки. Модели вычислений. Формальное определение алгоритма. Различные определения трудоемкости алгоритма. Теоремы иерархии (без доказательства).
2. Асимптотические обозначения (O, Ω, θ, o, ω) и их свойства. Алгоритмы типа «разделяй-и-властвуй». Основная теорема о рекуррентных оценках (нахождение асимптотики рекуррентности вида $T(n) = aT\left(\frac{n}{b}\right) + f(n)$). Дерево рекурсии. Линейный алгоритм нахождения медианы массива. Линейные рекуррентные последовательности.
3. Детерминированные и недетерминированные МТ. Класс *P*. Примеры языков из *P*: принадлежность слова регулярному языку; принадлежность слова КС-языку; системы линейных уравнений (полиномиальная реализация метода Гаусса). Классы *NP* и со-*NP*. Примеры языков из *NP*: выполнимость, простые числа; пересечение регулярных языков, заданных конечными автоматами; непланарные графы.
4. Полиномиальная сводимость. Сводимость по Карпу и по Куку (по Тьюрингу). Теорема Кука-Левина. Примеры полиномиально полных языков: выполнимость; протыкающее множество; 3-сочетание; максимальное 2-сочетание; вершинное покрытие; клика; хроматическое число; гамильтонов цикл; рюкзак; разбиение; максимальный разрез. *PSPACE*. Теорема Савича (без доказательства). Полные языки в *PSPACE*: истинность булевской формулы с кванторами; эквивалентность конечных автоматов; непустота пересечения последовательности ДКА.
5. Потоки и разрезы в сети. Теорема о максимальном потоке и минимальном разрезе. Понятие остаточного графа и увеличивающего пути. Алгоритм Форда-Фалкерсона для вычисления максимального потока и минимального разреза. Полиномиальная реализация алгоритма максимального потока. Задача о максимальном потоке минимальной стоимости. Обобщения потоковой сети (пропускные способности узлов и пр.).

Приложение потоковых алгоритмов: цепное разложение порядков (лемма Дилворта), задача о максимальном паросочетании в двудольном графе, задача о назначениях, расписание с прерываниями на идентичных процессорах.

6. Задача линейного программирования (ЛП). Основные понятия. Выпуклые многогранники. Теорема двойственности. Использование ЛП для точного и приближенного решения переборных задач комбинаторной оптимизации: задача о назначении; паросочетание; вершинное покрытие; коммивояжер.
7. Алгоритмы сортировки: пузырек; быстрая сортировка (quicksort); сортировка с помощью кучи; слияние; цифровая сортировка. Анализ трудоемкости алгоритма quicksort по наихудшему случаю и в среднем. Устойчивость алгоритма сортировки. Нижние оценки сортировки. Разрешающие деревья. Порядковые статистики.
8. Обобщенный алгоритм Евклида. 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. Схема RSA. Протокол Диффи-Хелмана. Рациональные приближения. Элементарные свойства цепных дробей. Хеш-таблицы. Разрешение коллизий с помощью цепочек. Хеш-функции (деление с остатком, умножение). Универсальные и *k*-универсальные хеш-функции.
9. Дискретное преобразование Фурье; алгоритм быстрого преобразования Фурье (БПФ); перемножение многочленов с помощью БПФ. Использование БПФ для распознавания образцов.
10. Алгоритмы на графах и порядках: поиск в ширину; поиск в глубину; определение двусвязных и/или сильно связных компонент; кратчайшие пути: алгоритмы Дейкстры, Флойда, Беллмана-Форда; транзитивное замыкание; топологическая сортировка; остовные леса (алгоритмы Прима и Краскала); паросочетания.
11. Методы решения переборных задач: динамическое программирование, шкалирование, ветви и границы, приближенные алгоритмы. ε-оптимальная процедура решения задачи о рюкзаке.
12. Классы *RP*, *BPP*, *ZPP*. Лемма Шварца. Вероятностные алгоритмы: проверка простоты, вычисление медианы массива, проверка полиномиальных тождеств, поиск паросочетаний в графах, алгоритм Каргера поиска минимального разреза.

ЗАДАНИЕ

Задание состоит из списка недельных заданий (указаны даты обсуждения на семинарах и/или сдачи соответствующих задач). Каждое недельное задание состоит из четырех-пяти обязательных задач и одной-двух дополнительных задач (дополнительные задачи выделены буквой «Д»).

Решение дополнительных задач не обязательно, но может быть полезно студентам, претендующим на более высокую оценку.

Обозначения

$\lfloor a \rfloor$ – наибольшее целое, не превосходящее число *a;*

$\lceil a \rceil$ – наименьшее целое, превосходящее *a;*

φ(*n*) – функция Эйлера, равная количеству натуральных чисел, меньших *n* и взаимно простых с ним. При этом полагают, что число 1 взаимно просто со всеми натуральными числами.

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

Пусть *f*(*n*) и *g*(*n*) – неотрицательные функции.

- *f*(*n*) = *O*(*g*(*n*)) означает, что порядок роста *g* при n → ∞ не меньше порядка роста *f*, т. е. ∃ $c > 0$, такое что при $n > n_0$, $f(n) \leq c\, g(n)$;
- *f*(*n*) = Ω(*g*(*n*)) ⇔ *g*(*n*) = *O*(*f*(*n*)) (порядок роста *f* не меньше порядка роста *g*);
- *f*(*n*) = *o*(*g*(*n*)), если $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 0$ (*f* имеет меньший порядок роста, чем *g*);
- *f*(*n*) = ω(*g*(*n*)) ⇔ *g*(*n*) = *o*(*f*(*n*)) (*f* имеет меньший порядок роста, чем *g*);
- *f*(*n*) = θ(*g*(*n*)), если *f*(*n*) = *O*(*g*(*n*)) и *f*(*n*) = Ω(*g*(*n*)) (порядки роста *f* и *g* одинаковы).

Задание на 1-ю неделю занятий. Разделы 1 и 2 программы. Оценки

1. Покажите, что $\sum_{i=1}^{n} \frac{1}{i} = \theta(log\ n)$.

2. Найдите θ-асимптотику C_{2n}^{n}.

3. Задача на применение «Основной теоремы о рекуррентных оценках» **[Кормен 1, § 4.3]**. Вы должны проверить, выполняются ли условия теоремы, получить оценки и сравнить их. Пока можно считать, что $n = 2^k$ или $n = 3^k$.

Укажите рекурсивную процедуру с наименьшей асимптотической сложностью:

- алгоритм, разбивающий задачу размера n на девять подзадач размера $\frac{n}{3}$ и использующий для этого $\theta(n^2 \log n)$ операций;
- алгоритм, разбивающий задачу размера n на шестнадцать подзадач размера $\frac{n}{4}$ и использующий для этого $\theta(n^2)$ операций;
- алгоритм, разбивающий задачу размера n на четыре подзадачи размера $\frac{n}{2}$ и использующий для этого $\theta\left(\frac{n^2\sqrt{n}}{log^2 n}\right)$ операций.

4. Найдите θ-асимптотику следующих рекуррентностей.

4.1. $T_1(n) = 2T_1\left(\frac{n}{2}\right) + n.$

4.2. $T_2(n) = 3T_2\left(\frac{n}{3}\right) + n^2.$

4.3. $T_3(n) = 4T_3\left(\frac{n}{2}\right) + \frac{n}{\log n}.$

Быстрое умножение чисел и матриц

5. Алгоритм быстрого умножения чисел (алгоритм Карацубы) подробно описан во многих источниках (см., например, **[АХУ]**). Ниже приведен пример работы этого рекурсивного алгоритма (стрелка после произведения указывает на вспомогательные произведения, которые требуется вычислить; рекурсия останавливается на двузначных числах):

1. $216_{10} \times 139_{10} = 11011001 \times 10001011 \rightarrow$
2. $1101 \times 1000, 1001 \times 1011, (1101 + 1001) \times (1000 + 1011) = 10110 \times 10011;$
3. $1101 \times 1000 \rightarrow 11 \times 10 = 110;\ 01 \times 00 = 0;\ (11 + 01) \times (10 + 00) = 100 \times 10 = 1000;$
4. $1101 \times 1000 = 1100000 + (1000 - 110 - 0) \times 100 + 0 = 1101000;$
5. $1001 \times 1011 \rightarrow 10 \times 10 = 100,\ 01 \times 11 = 11, (10 + 01) \times (10 + 11) = 11 \times 101 = 1111;$
6. $1001 \times 1011 = 1000000 + (1111 - 100 - 11) \times 100 + 11 = 1100011;$
7. $010110 \times 010011 \rightarrow 010 \times 010 = 100, 110 \times 011, (010 + 110) \times (010 + 011) = 1000 \times 0101 = 101000;$
8. $0110 \times 0011 \rightarrow 01 \times 00 = 0, 10 \times 11 = 110, (1 + 10) \times (0 + 11) = 11 \times 11 = 1001;$
9. $0110 \times 0011 = 0 + (1001 - 0 - 110) \times 100 + 110 = 10010;$
10. $010110 \times 010011 = 100000000 + (101000 - 10010 - 100) \times 1000 + 10010 = 110100010;$
11. $11011001 \times 10001011 = 110100000000000 + (110100010 - 1101000 - 1100011) \times 10000 + 1100011 = 111010111010011 = 30163_{10}.$

Получим рекуррентность для оценки полного числа всех элементарных битовых арифметических операций алгоритма Карацубы (вы должны самостоятельно сформулировать понятие «элементарная битовая арифметическая операция»). Считаем, что $n = 2^k$. Обозначим Add(m) — число элементарных битовых арифметический операций, требуемых для сложения двух m-битовых чисел. Обозначим Mult(m) — число элементарных битовых арифметических операций, требуемых для умножения двух m-битовых чисел методом Карацубы. Запишем соотношения между Mult(.) и Add(.), вытекающие из алгоритма $\text{Mult}(2m) \le 3\ \text{Mult}(m+1) + O\big(\text{Add}(m)\big)$. Обычное школьное сложение в столбик дает оценку для $\text{Add}(\text{m}) = O(\text{m})$, и мы получаем рекуррентное соотношение на Mult(.).

Приведите подробный асимптотический анализ рекуррентности.

Основная трудность заключается в оценке прибавления единицы к аргументу во втором члене: нужно привести формальные аргументы, показывающие, почему этой поправкой можно пренебречь.

Д1. В этой задаче нужно применить приемы вычисления рекуррентностей, изложенные в книге **[Кормен 1]** или **[Кормен 2]**, а также провести анализ, связанный с использованием функций $\lfloor \cdot \rfloor$ в рекуррентности.

Оцените высоту дерева рекурсивных вызовов рекуррентности $T(n) = T\big(n - \lfloor \sqrt{n} \rfloor\big) + T(\lfloor \sqrt{n} \rfloor) + \theta(n)$ как можно точнее.

Задание на 2-ю неделю занятий. Разделы 2 и 3 программы

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

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

В 1969 году Ф. Штрассен (V. Strassen) открыл следующий алгоритм: пусть $p_1 = a(f-h)$; $p_2 = (a+b)h$; $p_3 = (c+d)e$; $p_4 = d(g-e)$; $p_5 = (a+d)(e+h)$; $p_6 = (b-d)(g+h)$; $p_7 = (a-c)(e+f)$.
Тогда

$$ae + bg = p_5 + p_4 - p_2 + p_6 \quad af + bh = p_1 + p_2$$
$$cf + dh = p_5 + p_1 - p_3 - p_7 \quad ce + dg = p_3 + p_4$$

Таким образом, нам требуется 7 умножений и 18 сложений. На первый взгляд, мы ничего не выиграли: умножений стало меньше, но сложений больше, но тут начинается маленькое чудо. Обратим внимание на то, формулы Штрассена справедливы и тогда, когда переменные некоммутативные, и поэтому можно рекурсивно запустить алгоритм (подробности можно прочитать в **[Кормен 2, 28.2]**), 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. Поскольку сложение матриц требует $O(n^2)$, то получится такая рекуррентная оценка трудоемкости алгоритма перемножения матриц: $T(n) = 7T\left(\frac{n}{2}\right) + O(n^2)$.

Оцените трудоемкость алгоритма Штрассена, *используя дерево рекурсии* (можно считать, что $n = 2^k$).

7. Дано n точек плоскости. Предложите как можно более быструю процедуру нахождения круга минимального радиуса с центром в начале координат, содержащего не менее половины точек. (Считаем, что арифметические операции и сравнения выполняются за единицу времени.)

8. Рассмотрим рекурсивный алгоритм вычисления медианы массива, который отличается от стандартного **([Кормен 1, 10.3]** или **[Кормен 2, 9.3]**) тем, что массив делится не на «пятерки», а на «девятки» элементов.

8.1. Получите рекуррентное соотношение для трудоемкости этого алгоритма.

8.2. Оцените трудоемкость алгоритма с помощью дерева рекурсий.

9. Пусть G — язык в алфавите {0,1,2}, состоящий из всех слов, в которых соседние символы отличаются (как числа) не более чем на 1, например, 210 ∈ G, 201 ∉ G. Найдите рекуррентное соотношение, которому удовлетворяет последовательность $\{g_n,\ n = 1, 2,\ldots\}$, где g_n равно числу слов длины n в G, и оцените g_{2014}.

Д2.
Д2.1. Сравните по величине 2014-е члены рекуррентных последовательностей: $a_0 = b_0 = 0;\ a_{n+1} = {a_n}^2 + 5,\ b_{n+1} = {a_n}^2 + 2^n$.
Д2.2. Найдите как можно более точную асимптотику последовательности a_n.

Задание на 3-ю неделю занятий. Разделы 2 и 3 программы.
Класс *P*. Полиномиальные алгоритмы

11. Язык $L_{m\times 2014-incons} = \{(A, b)\}$ состоит из кодировок всех несовместных систем линейных уравнений $Ax = b$ с целыми коэффициентами и 2014 неизвестными, в которых максимальный модуль целых коэффициентов матрицы A и вектора b («высота» системы)

не превышает 10^5. Длиной записи считаем суммарную длину двоичных записей всех коэффициентов системы.

11.1. Укажите два слова, принадлежащие языку, и два слова, не принадлежащие языку.

11.2. Покажите, что $L_{m\times 2014-incons} \in P$.

Подсказка. Иначе говоря, нужно показать, что существует машина Тьюринга M со следующими свойствами. На вход M поступает слово, содержащее двоичные записи коэффициентов системы. M останавливается через полиномиальное по длине входа число шагов и печатает ответ 1, если система несовместна, или печатает 0, если система совместна.

Комментарий. Не рекомендуется бодро ссылаться, скажем, на теорему Кронеккера-Капелли, не указывая, как вычислять ранг матрицы за полиномиальное по длине записи время, поскольку в известном со школы методе исключений длина записи промежуточных вычислений либо игнорируется, либо излагается в других терминах.

Снимем теперь ограничение на размер и высоту системы, т. е. рассмотрим систему линейных уравнений $Ax = b$ с целыми коэффициентами, имеющую m уравнений и n неизвестных, причем максимальный модуль целых коэффициентов матрицы A и вектора b равен h.

11.3. Оцените сверху числители и знаменатели чисел, которые могут возникнуть при непосредственном применении алгоритма Гаусса.

Покажем на примере этой простой задачи критическую важность выбора модели вычислений. Допустим, что в нашем распоряжении имеется PAM, т. е. равнодоступная адресная машина (random access machine — RAM)[1], в которой арифметические операции над очень большими числами, например, имеющими экспоненциальное по входу число битов, выполняются за единицу времени. В этом случае можно, например, не учитывать длину записи промежуточных вычислений в методе Гаусса, и его трудоемкость будет в точности такая, как об этом пишут в учебниках вычислительной математики.

Построим алгоритм, выполняющий полиномиальное по длине записи натурального n количество арифметических операций над экспоненциально длинными числами, который решает задачу факторизации, т. е. находит нетривиальные делители числа n. Подобная PAM может в принципе вскрывать большинство современных шифров, в которых факторизация является одним из важнейших примитивов. Идея процедуры такова. Рассмотрим функцию натурального аргумента $f_n(a) = \text{НОД}(a!, n)$. Следующие три утверждения непосредственно следуют из определения:

1) $f_n(1) = 1,\ f_n(n) = n$;
2) $f_n(a)$ не убывает при $1 \le a \le n$;
3) число n составное тогда и только тогда, когда существует такое число $1 < a_0 < n$, что $f_n(a_0)>1$.

Итак, нетривиальный делитель a_0 отвечает первому «скачку» $f_n(a)$, а для того, чтобы найти его, нужно воспользоваться обычным двоичным поиском. Осталось научиться вычислять $f_n(a)$, используя $poly(\log n)$ арифметических операций [возможно, над очень

[1] Грубо говоря, PAM-программа − это программа на BASIC'е. Здесь полезно подумать, как формально определить PAM и чем она операционально отличается от MT. Описание PAM можно найти, например, в книге **[АХУ]** или посмотреть в сети.

большими числами]. Можно ограничиться $O(\log n)$ операциями, но мы рассмотрим более простой и более трудоемкий способ. Сначала решим следующую задачу.

12.

12.1. Покажите, что язык $L_{ind} = \{a, b, c, d$ — натуральные числа, такие что $a^b = c \ mod \ d\}$ принадлежит P.

12.2. Укажите два слова, принадлежащие языку, и два слова, не принадлежащие языку.

Возвращаемся к задаче вычисления скачка $f_n(a)$. По аналогии с возведением в степень в предыдущей задаче факториал инкрементально пересчитывать, если воспользоваться следующими формулами, прямо следующими из определения: $(2и + 1)! = (2и + 1)(2и)!$ и $(2b!) = (b!)^2 C_{2b}^b$. Для вычисления C_{2b}^b применим еще один трюк. Пусть $2b < n$. Тогда можно, используя $poly(\log n)$ арифметических операций, получить число C_{2b}^b, сначала вычисляя степень $A = (2^n + 1)^{2b}$, а потом выделяя нужные биты A. Так можно получить алгоритм, требующий для факторизации $O(log^3 n)$ операций, который, конечно, не будет полиномиальным.

13. Покажите, что в класс P входит язык $L_{powers} = \{1, 10^{2014}, \dots\}$, состоящий из двоичных записей 2014-х степеней натуральных чисел.

14. Покажите, что класс P как множество языков замкнут относительно итерации (операции Клини).

В изображенных ниже таблицах используются следующие обозначения:
A — конечный автомат (КА);
NA — недетерминированный КА (НКА);
DA — детерминированный КА (ДКА);
R — регулярное выражение (РВ);
O — магазинный автомат (МП);
N — магазинный автомат, принимающий по пустому стеку;
F — магазинный автомат, принимающий по финальному состоянию;
G —. контекстно- свободная грамматика (КСГ);
M — машина Тьюринга.
L(•), где вместо «•» можно подставить один из перечисленных выше объектов, обозначает класс языков, принимаемых (порождаемых) объектами указанного класса. Например, L(G) означает класс всех КС-языков, причем описание каждого языка дается соответствующей КСГ. Будем считать, что объекты задаются своими стандартными описаниями, например, НКА кодируется его диаграммой.
Если специально не оговорено, то предполагается, что все языки заданы в двоичном алфавите $\{0,1\}$.
В таблицах ниже строки отвечают предикатам, а столбцы — классам языков, которые задаются указанными объектами. Рассматриваются следующие предикаты[2]:

[2] 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

$L(\bullet) = \emptyset$? — язык пуст;
$|L(\bullet)| = \infty$? — язык бесконечен;
$w \in (\notin) L(\bullet)$? — слово «w» принадлежит (не принадлежит) языку;
$L(\bullet_1) = L(\bullet_2)$? — языки, порождаемые объектами $(\bullet_1)$ и $(\bullet_2)$, эквивалентны.

Нашей целью является оценка сложности получаемых «задач», т. е. оценка сложности вычисления указанных предикатов на соответствующих классах языков. Например, ячейке (2,2) табл. 1 отвечает задача проверки непустоты языка, заданного ДКА, а ячейке (2,4) табл. 2 — задача проверки эквивалентности языков, заданных магазинными автоматами. Обе таблицы будут заполняться в несколько проходов, и в нулевом приближении полезно прикинуть, какие из указанных задач разрешимы, а какие — нет. Уточнением таблиц мы будем заниматься на протяжении курса.

Таблица 1

	DA	NA	R	G	N	F
$L(\bullet) = \emptyset$?						
$\|L(\bullet)\| = \infty$?						
$w \in L(\bullet)$?						
$w \notin L(\bullet)$?						

Таблица 2

	DA и DA	DA и NA	DA и R	O и O
$L(\bullet_1) = L(\bullet_2)$?				
$L(\bullet_1) \neq L(\bullet_2)$?				

15. Покажите, что в столбцах табл. 1, отвечающих, соответственно DA, NA, R и G и в столбце табл. 2, отвечающему [DA и DA], можно поставить знак «*P*»; это означает, что соответствующие задачи разрешимы за полиномиальное по входу время.

Д3.

Д3.1. Постройте полиномиальный по входу алгоритм для задачи линейного программирования на плоскости:

$$a_0x + b_0y \to min$$
$$a_1x + b_1y = c_1$$
$$\dots$$
$$a_kx + b_ky = c_k.$$

несимметричны: Каждый понимает, что ситуация, когда программа остановилась или когда она еще работает, существенно отличаются, и поэтому, например, есть предикаты, имеющие алгоритм проверки выполнимости и не имеющие алгоритма проверки невыполнимости и наоборот. Знаете ли вы такие предикаты?

Здесь все коэффициенты a_i, b_i, c_i — целые числа, по абсолютной величине меньшие h (высоты системы). Входом задачи можно считать $\max\{k, log\ h\}$.

Д3.2. Будем теперь считать, что любая арифметическая операция над коэффициентами и операция сравнения занимают единицу времени. Постройте линейный $O(k)$-алгоритм для рассмотренной задачи.

Д4. Покажите, в столбцах Таблицы 1, отвечающих, соответственно, N и F, можно поставить знак «*P*».

Комментарий. Итак, все задачи, указанные в Таблице 1, разрешимы за полиномиальное по входу время.

Задание на 4-ю неделю занятий. Разделы 3 и 4 программы. Класс *P*, полиномиальный вариант метода Гаусса

16. В задаче **11** при оценке трудоемкости метода исключений Гаусса мы столкнулись с трудностью, поскольку из решения задачи **11.3** следует, что при прямом использовании алгоритма Гаусса промежуточные результаты могут в принципе расти дважды экспоненциально и потому, в частности, «наивный» метод исключений не является полиномиальным по входу в битовой арифметике. Но оказывается, что метод Гаусса можно модифицировать так, что получится полиномиальный алгоритм. Модификация заключается в эмуляции *рациональной арифметики*. Для этого каждый (рациональный) коэффициент $\frac{p}{q}$ представляется парой (p',q') взаимно простых чисел, таких что $\frac{p}{q} = \frac{p'}{q'}$. 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 ($7 \times 6 + 5 \times 10 = 92$, $6 \times 10 = 60$), определении НОД(92, 60) = 4 и записи ответа (23, 15).

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

Рассмотрим систему линейных уравнений $Ax=b$, где $A - n \times n$ матрица и $b - n \times 1$ вектор-столбец. Без ограничения общности будем считать, что все ведущие элементы в методе Гаусса расположены на главной диагонали. Обозначим $(a_{ij}^{(k)})$ матрицу, полученную после k-о исключения. Также обозначим $d_1, \dots, d_n$ элементы главной диагонали результирующей верхнетреугольной матрицы, так что $d_i = a_{ii}^{(n)}, i = 1, \dots, n$. Пусть $D^{(k)} - k \times k$-подматрица, образованная первыми k столбцами и первыми k строками исходной матрицы системы, а $D_{ij}^{(k)} - (k+1) \times (k+1)$-подматрица, образованная первыми k столбцами и столбцом i, и первыми k строками и строкой j, матрицы, полученной после k-о исключения. Пусть $d_{ij}^{(k)} = \det(D_{ij}^{(k)})$. По определению, $\det(D^{(k)}) = d_{kk}^{(k-1)}$. **Ключом является следующая формула**: $a_{ij}^{(k)} = \frac{a_{ij}^{(k)}}{\det(D^{(k)})}$, поскольку, в соответствии с процедурой исключений $d_{ij}^{(k)} =$

$d_1 \cdots d_k a_{ij}^{(k)}$ и $\det\left(D^{(k)}\right) = d_1 \cdots d_k$. Таким образом, можно все время работать с дробями, числители и знаменатели которых являются минорами исходной матрицы.

16.1. Проведите вычисления модифицированного алгоритма Гаусса для матрицы $A = \begin{matrix} 1 & 0 & 0 \\ 3 & 4 & 0 \\ 9 & 3 & 1 \end{matrix}$ и вектора $b^t = (4 \quad 16 \quad 64)$.

Внимание! Вычисления должны проводиться строго по описанию алгоритма: начинаем с элемента a_{11}, «чистим» первый столбец; затем берем в качестве ведущего первый ненулевой элемент второй строки и «чистим» соответствующий столбец и т. д. Алгоритм априори не «знает», что матрица *A* нижнетреугольная.

16.2. По результатам вычислений предыдущего пункта численно проверьте равенство $d_{33}^{(2)} = d_1 d_2 a_{33}^{(2)}$.

Классы *NP* и *co-NP*

17. Покажите, что следующие языки принадлежат *NP* (постройте соответствующие сертификаты «*y*» и проверочные предикаты *R*(*x*, *y*).

17.1. Язык совместных систем линейных уравнений с целыми коэффициентами от 2014 неизвестных.

17.2. Язык совместных систем линейных неравенств с целыми коэффициентами от 2014 неизвестных.

17.3. Составные числа, не содержащие в двоичной записи подслова 10101.

Разберем нетривиальный пример. Просто понять, что в *NP* лежит язык составных чисел: $A = \{1,4,6,8.10, \dots\}$ (сертификатом служат предъявляемые сомножители). Но оказывается, что в *NP* лежит и дополнительный язык $B = \boldsymbol{N} \setminus A = \{2,3,5, \dots\}$ простых чисел [3] Полиномиальный сертификат устроен хитро. Как мы знаем из курса дискретного анализа, $p \in B \Leftrightarrow \exists\, g\colon \{g^i mod\, p, i = 0,1 \dots\} = \{1,2, \dots, p-1\}$ (указано равенство множеств). Поскольку длина записи числа *p* составляет *log p*, то длина *NP*-сертификата должна быть *poly*(*l*og *p*). И если быстро возводить числа в степень *mod p* мы еще умеем (каким образом?), то все равно массив $\{g^i mod\, p\}$ слишком длинный (экспоненциальный по длине записи). Но, как мы помним, вычет *g* с нужными свойствами существует тогда и только тогда, когда выполнено: $g^{\frac{p-1}{p_1}} \neq 1\ mod\ p, \dots, g^{\frac{p-1}{p_k}} \neq 1\ mod\ p$, где $p_1, \dots, p_k$— это все простые делители числа $p - 1 = {p_1}^{l_1} \dots {p_k}^{l_k}$. Число проверок действительно уменьшилось и стало полиномиальным (их заведомо не больше *log p*), но, кажется, что мы поменяли шило на мыло: нам ведь нужно решить **исходную задачу построения сертификата простоты** для всех $p_1, \dots, p_k$. Хитрость заключается в том, что нужно применить ту же идею рекурсивно, поскольку длина сертификатов для всех p_1 сильно уменьшилась! Фактически сертификатом будет дерево с нужными пометками в вершинах, и для завершения

[3] В 2002 году появилась сенсационная работа, показывающая, что $B \in P$. Если вы будете ссылаться на ее результаты, то ОБЯЗАНЫ привести доказательство.

доказательства нужно показать, что суммарная длина всех участвующих в описании дерева компонентов останется полиномиальной по $\log p$ (это предлагается сделать в **Д6**).

18. Постройте *NP*-сертификат простоты для числа $p = 3361$. Число «22» является первообразным корнем mod p. Простыми в рекурсивном построении считаются только числа 2, 3, 5 (они сами являются своими сертификатами).

19. Покажите, что класс *NP* замкнут относительно итерации (операции Клини). Укажите, как построить для результирующего языка соответствующий сертификат «y» и проверочный предикат $R(x, y)$.

Рассмотрим еще один нетривиальный пример.
Граф $G(V, E)$ называется *планарным*, если его можно вложить в плоскость, т. е. существует отображение $f: G \to R^2$, посылающее вершины V в различные точки плоскости, а ребра E – в кривые (которые можно считать ломаными), соединяющие соответствующие точки-вершины, при этом кривые-ребра могут пересекаться только по вершинам.

20. Пусть L_{planar} — это язык планарных графов (как обычно, считаем, что графы задаются списком ребер или матрицей смежности). Покажите, что $L_{planar} \in co - NP$.

Комментарий. Как известно, на плоскости нельзя нарисовать без пересечения ребер (вложить) ни граф $\boldsymbol{K_5}$ (полный пятивершинник), ни граф $\boldsymbol{K_{3,3}}$ (три дома, три колодца), и знаменитая теорема Куратовского (или Понтрягина-Куратовского) утверждает, что граф планарен тогда и только тогда, когда в нем нет подграфа, полученного из $\boldsymbol{K_5}$ или $\boldsymbol{K_{3,3}}$ посредством подразбиения ребер.

Д5. Покажите, что *NP* принадлежит язык из пункта **17.2** при дополнительном предположении, что переменные $\{x_i\}$ целочисленные. Если эта задача представляет трудности, то разберите случай, когда неизвестных две.

Д6. Покажите, что длина *NP*-сертификата простоты, описанного в задаче **18**, равна $O(\log p)$.

Д7. 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. Скажем, $\boldsymbol{K_5}$ и $\boldsymbol{K_{3,3}}$ можно вложить в тор, поскольку на плоскости удается изобразить все ребра этих графов, кроме одного, которое можно пустить по ручке (вдоль параллели тора). Но удастся ли такой трюк с графом $\boldsymbol{K_{3,6}}$ (число домов осталось прежним, но добавлено еще три колодца)?

Д7.1. Докажите или опровергните, что граф $K_{3,6}$ является торическим. Тор удобно изображать прямоугольником с отождествленными противоположными сторонами.
Д7.2. Покажите, что класс торических графов принадлежит *NP*.

Задание на 5-ю неделю занятий. Разделы 3 и 4 программы. Полиномиальная сводимость

21. Язык ГЦ состоит из всех графов, имеющих гамильтонов цикл (несамопересекающийся путь, проходящий через все вершины графа). Язык ГП состоит из всех графов, имеющих гамильтонов путь. Постройте ***явные*** полиномиальные сводимости ГП к ГЦ и наоборот. Графы кодируются, например, их матрицей смежности.
Комментарий. Это очень хорошее упражнение для того, чтобы понять определение полиномиальной сводимости.

NP-полные языки

Самый известный *NP*-полный язык — это, конечно, ВЫПОЛНИМОСТЬ, который состоит из кодировок всех выполнимых булевых формул. Иначе говоря, для каждой формулы 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 трех 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 *все литералы в каждом дизъюнкте 3-КНФ различны*. Но от этого требования можно и отказаться, если окажется проще строить какие-то сводимости, т. е. рассмотреть более широкий полный язык, в котором литералы в дизъюнктах могут повторяться и в каждый дизъюнкт входит не более трех литералов. Такой трактовки языка 3-ВЫПОЛНИМОСТЬ мы и будем придерживаться в этом задании. Тогда при часто используемом преобразовании 3-КНФ в РОВНО-3-КНФ можно просто дополнить дизъюнкт нужным числом литералов. Например, дизъюнкт $\neg x_2 \vee x_3$ переписывается в эквивалентном виде $\neg x_2 \vee x_3 \vee x_3$ или $\neg x_2 \vee x_3 \vee \neg x_2$. Другое дело, что некоторые сводимости при таком понимании 3-КНФ, возможно, перестанут выполняться, и тогда нужно уточнить и/или изменить сами сводимости.

Приведем несколько примеров *NP*-полных языков.

ПРОТЫКАЮЩЕЕ МНОЖЕСТВО. Дано семейство конечных множеств $\{A_1, \dots A_n\}$ 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.

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*.

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 $A' \subseteq A$, что $\Sigma_{a\in A'}\ s(a) = \Sigma_{\ a\in A\setminus A'} s(a)$.

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.

22. 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, причем литералы могут повторяться, в РОВНО-3-КНФ $\widetilde{\varphi}$, в которой в каждый дизъюнкт входит РОВНО три неповторяющихся литерала. При этом φ должна быть выполнима тогда и только тогда, когда выполнима $\widetilde{\varphi}$. Иными словами, постройте полиномиальную сводимость языка 3-ВЫПОЛНИМОСТЬ к языку 3-ВЫПОЛНИМОСТЬ (по Кормену).

Теперь зафиксируем выполнимую КНФ $\psi(x_1, x_2,) = (\neg x_1 \vee x_2)$ [зависящую от двух переменных и имеющую 1 дизъюнкт] и невыполнимую КНФ $\chi(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_3$ [зависящую от трех переменных и имеющую 5 дизъюнктов]. Везде ниже мы будем иллюстрировать сводимости, используя именно эти КНФ.

23. Сводимость языка ВЫПОЛНИМОСТЬ **к языку** ПРОТЫКАЮЩЕЕ МНОЖЕСТВО**.** Конструкция такова. Пусть $\varphi(x_1, \ldots x_n)$ КНФ. Построим по КНФ семейство подмножеств A_φ

базового множества $\{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$. Во-первых, включим в A_φ n подмножеств вида $A_i = \{x_i, \neg x_i\}$, $i=1,\ldots,n$. Во-вторых, для каждого дизъюнкта C, входящего в φ, добавим к A_φ 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 φ выполнима тогда и только тогда, когда A_φ имеет протыкающее множество мощности n. Обоснование легко получить, если решить две следующие задачи.

23.1. Укажите для семейства A_ψ соответствующее *двухэлементное* протыкающее множество.

23.2. Докажите, что мощность любого протыкающего множества для семейства A_χ больше трех.

Если использовать полный язык 3-ВЫПОЛНИМОСТЬ, то из построенной сводимости следует, что язык остается *NP*-полным, даже если все A_i имеют не более трех элементов. Но оказывается, что язык остается *NP*-полным, даже если все A_i двухэлементные. Если отождествить эти пары элементов с ребрами некоторого графа, то соответствующий язык известен как ВЕРШИННОЕ ПОКРЫТИЕ: даны неориентированный граф $G=(V,E)$ и натуральное число k. 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[4]. Будем считать, что исходная КНФ дополнена до РОВНО-3-КНФ и в каждый ее дизъюнкт входит ровно три литерала.

Сводимость языка РОВНО-3-КНФ-ВЫПОЛНИМОСТЬ **к языку** ВЕРШИННОЕ ПОКРЫТИЕ. Построим по РОВНО-3-КНФ $\varphi(x_1,\ldots,x_n)$ 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 дизъюнктов, 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 $n+2m$. Обоснование может быть построено, если решить две следующие задачи.

23.3. Укажите для графа G_ψ соответствующее $(n_{new}(\psi) + 2m_{new}(\psi))$-вершинное покрытие.

23.4. Докажите, что мощность любого вершинного покрытия для графа G_χ больше $(n_{new}(\chi) + 2m_{new}(\chi))$.

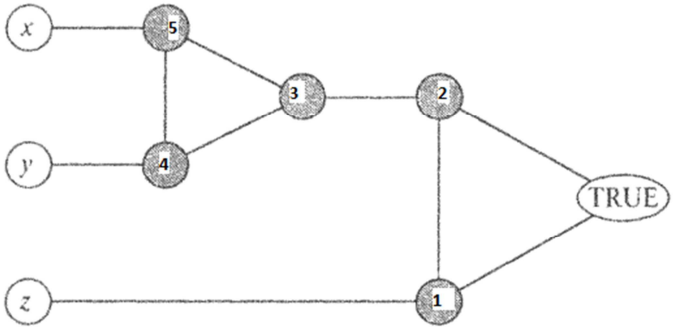
---

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

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

Задание на 6-ю неделю занятий. Разделы 3 и 4 программы. *NP*-полные языки

Язык 3-COLOR состоит из кодировок всех неориентированных графов, вершины которых можно окрасить тремя цветами так, чтобы смежные вершины имели разные цвета.
Сводимость языка 3-ВЫПОЛНИМОСТЬ **к языку** 3-COLOR**.** Сводимость основана на свойствах вспомогательного графа, с помощью которого можно по любой 3-КНФ φ построить граф W_φ (V,E), допускающий раскраску в три цвета тогда и только тогда, когда φ выполнима. В графе каждой переменной и ее отрицанию отвечает по вершине, а каждому дизъюнкту отвечает пять вершин. Кроме того, есть три выделенные вершины R, T и F. В G есть ребра двух типов. Литеральные ребра образуют треугольник на специальных вершинах и на всех тройках вершин вида x_i ,$\neg x_i$, R. Дизъюнктные ребра строятся по каждому дизъюнкту $x \vee y \vee z$ из φ, как показано на рисунке (пять вспомогательных дизъюнктных вершин закрашены).



24. Вычислите образ ψ при указанной полиномиальной сводимости (конечно, ψ сначала нужно преобразовать в 3-КНФ) и укажите 3-раскраску построенного графа $W_\psi(V,E)$.

25. Сводимость языка РОВНО-3-КНФ **к языку** КЛИКА **([Кормен 1, § 36.5.1] или [Кормен 2, § 34.5.1]).** По любой РОВНО-3-КНФ $\varphi(x_1,\dots,x_n)$ с m дизъюнктами строится граф Q_φ на $3m$ вершинах, в котором имеется клика размера m тогда и только тогда, когда $\varphi(x_1,\dots,x_n)$ выполнима. 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, а затем определить параметры s и t.
25.1. Укажите для графа Q_ψ соответствующую s-клику.

25.2. Докажите, что мощность любой клики в графе Q_χ меньше *t*.

26. Сводимость языка РОВНО-3-КНФ-ВЫПОЛНИМОСТЬ **к языку** max-2-ВЫПОЛНИМОСТЬ. Для любой 3-КНФ $\alpha = \bigwedge_{i=1}^{n}(a_i \vee b_i \vee c_i)$, 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: $L_i = \{a_i, b_i, c_i, d_i, \neg a_i \vee \neg b_i, \neg a_i \vee \neg c_i, \neg b_i \vee \neg c_i, a_i \vee \neg d_i, b_i \vee \neg d_i, c_i \vee \neg d_i\}$, где $d_i,\ i = 1, \dots, n$ — это новые логические переменные. Осталось проверить, что если *i*-я дизъюнкция выполнена [в 3-КНФ], то можно так подобрать значение переменной d_i, что не менее q дизъюнкций в L_i будут выполнены. А если *i*-я дизъюнкция не выполнена [в 3-КНФ], то при любом значении переменной d_i, меньше q дизъюнкций в L_i будут выполнены. 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 2-дизъюнктов будет выполнено.

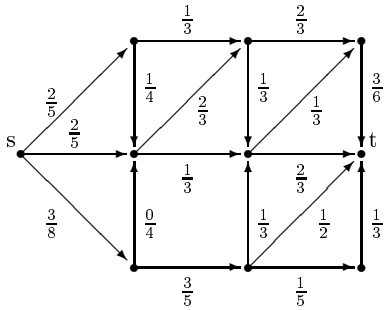
26.1. Преобразуйте ψ в РОВНО-3-КНФ [в которой образовалось k 3-дизъюнктов] и вычислите ее образ $\widetilde{\psi}$ при указанной полиномиальной сводимости, указав пороговое значение kq.

26.2. Укажите какой-нибудь набор значений логических переменных, при которых в $\widetilde{\psi}$ выполнено $\geq kq$ дизъюнктов.

27. Покажите, что если язык ГАМИЛЬТОНОВ ГРАФ $\in P$, то за полиномиальное время можно не только определить, что граф гамильтонов, но и найти в нем какой-нибудь гамильтонов цикл (если он существует).

Задание на 7-ю неделю занятий. Раздел 5 программы.
Потоки и разрезы в сетях

28. На рисунке изображен потоковый граф (метка $\frac{n}{k}$ на ребре означает поток и пропускную способность, соответственно).



28.1. Чему равен поток *f*?

28.2. Изобразите остаточный граф, соответствующий потоку *f*.

28.3. Максимален ли поток *f*?

В следующих двух пунктах нужно по шагам применить алгоритм Форда-Фалкерсона. При отсутствии алгоритма (например, если увеличивающие пути находятся методом «внимательного разглядывания» потокового графа или если разрез просто отгадывается) задача не оценивается. Метод остаточных графов из книг **[Кормен 1]** или **[Кормен 2]** аналогичен оригинальному методу пометок Форда-Фалкерсона, изложенному в их книге.

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

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

29. Покажите на примере конкретной сети, что алгоритм Форда-Фалкерсона не является полиномиальным.

30. Покажите, как найти *максимальное паросочетание* в двудольном графе, используя потоковый алгоритм в соответствующей сети.

31. Нужно распределить подпрограммы по двум различным процессорам (I и II) так, чтобы минимизировать общее время выполнения и обмена данных между процессорами. Пусть α_i (β_i) – время выполнения *i*-й подпрограммы на процессоре I (соответственно, на процессоре II), а γ_{pq} – время обмена данных между процессорами, если *p*-я подпрограмма выполняется на процессоре I, а *q*-я на процессоре II. Если обе подпрограммы выполняются на одном процессоре, то считается, что временем обмена данных можно пренебречь. Требуется минимизировать суммарное время выполнения подпрограмм и обмена данных между процессорами, т. е. если A (B) – это множество номеров подпрограмм, выполняющихся на процессоре I (соответственно, на процессоре II), то нужно найти минимум величины $\Sigma_{i\in A}\alpha_i + \Sigma_{i\in B}\beta_i + \Sigma_{i\in A, j\in B}\gamma_{ij} \to min$:

31.1. Покажите, как свести эту задачу к поиску минимального разреза в соответствующей сети.

31.2. Используя алгоритм Форда-Фалкерсона, найдите оптимальное распределение трех подпрограмм по процессорам.

Данные приводятся в таблице: во втором столбце указаны α_i, в третьем β_i, а последние три столбца содержат $\gamma_{ij}, 1 \le i, j \le 3$.

	α_i	β_i	прогр. 1	прогр. 2	прогр. 3
прогр. 1	3	∞		1	1
прогр. 2	3	1	4		2
прогр. 3	1	3	∞	1	

Промежуточный тест (темы: асимптотические оценки; алгоритмы типа разделяй-и-властвуй; линейный алгоритм медианы; классы P, NP, co-NP; полиномиальная сводимость; NP-полнота; потоки)

Задание на 8-ю неделю занятий. Разделы 5 и 6 программы. Линейное программирование

32. 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: $\max_{(a,b,c)\in \boldsymbol{R}^3} \min_{1\le i\le 7} |ax_i + by_i - c|$, здесь $(x_i, y_i)-$ координаты точек.

Комментарий. На лабораторных по физике 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.

33. Многогранник $P_\varepsilon \subset \boldsymbol{R}^3$ 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 $\varepsilon \in \left(0, \frac{1}{2}\right)$. 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.

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

$$A^\top p < 0, \qquad p \in R^m.$$
$$Ay = 0,\ y \ge 0, y \in R^n,\ y \ne 0. \ \textbf{(*)}$$

35. Пусть $H = \{y \in R^n \mid Ay = 0\}$ — векторное подпространство и $H^\perp$ — его ортогональное дополнение. Назовем вектор из R^n положительным, если все его координаты положительны. Альтернатива утверждает, что однородная система (*) несовместна тогда и только тогда, когда $H^\perp$ пересекает внутренность неотрицательного ортанта R^m_+. Иными словами, доказательством (сертификатом) того, что однородная система (*) 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*** полиноминально сводится к проверке совместности (*), и поэтому любую полиномиальную процедуру нахождения

сертификата «*а*» можно в принципе конвертировать в полиномиальный алгоритм линейного программирования.

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

36. Докажите, что для любого k в $\boldsymbol{R}^4$ существует выпуклый многогранник, имеющий k вершин (крайних точек), причем любая пара из них соединена ребром.

37. 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 черной.

Задание на 9-ю неделю занятий. Раздел 7 программы. Сортировка

38. 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 неочевиден, в частности, не проходит традиционный аргумент «по размеру входа», поскольку в $n/2$ 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 (§9.1 из книги **[Кормен 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 $\leq$, а другое $-$ $>$. 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 дается 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.*

38.1. Докажите импликацию: $\mathcal{B} \Longrightarrow \mathcal{A}$.

38.2. Докажите утверждение $\mathcal{B}$.

Теперь дадим другое доказательство этой нижней оценки. 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», определенную на конфигурациях: *ƒ*[(*a,b,c,d*)] = *a*+*c*.

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

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

39.1. Покажите, что любое разрешающее дерево поиска медианы позволяет также восстановить индексы всех элементов, б´ольших медианы, и всех элементов, м´еньших медианы.

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

39.2. 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 $\geq 2^{\frac{n}{2}-1}$ 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: $(2 + \varepsilon)n \leq T(n) \leq 2.95n$.

40. Инверсией в последовательности $A = \{a_1, \dots, a_n\}$ называется пара индексов (i,j), таких что $i < j$ и $a_i > a_j$. Постройте $O(n \log n)$-алгоритм подсчета числа инверсий в *A*.

Д8. [задача 10-1-2 из книги Кормен 1] 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 «неоптимальны». 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. $f(\text{Init}) - k\delta = f(\text{Final})$.
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.

Д9.1. 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.

Д9.2. Покажите, что любая детерминированная процедура подбора ключей требует $\Omega(n \log n)$ шагов.

Задание на 10-ю неделю занятий. Раздел 8 программы. Теоретико-числовые алгоритмы

41. В задаче **9** была определена последовательность $\{g_n,\ n = 0,1,\ldots\}$ и получено рекуррентное соотношение, которому она удовлетворяет. Оценим трудоемкость нескольких процедур вычисления элементов g_n по простому модулю p.

41.1. Оцените трудоемкость алгоритма прямого вычисления g_n по рекуррентной формуле. Оцените количество операций при вычислении $A=g_{10000}\ mod\ 19$.

41.2. Докажите, что последовательность $\{g_n\}$ периодична по любому модулю. Оцените ее период для mod 19 и найдите трудоемкость

алгоритма вычисления A (сложность нахождения периода плюс сложность вычисления A) этим способом.

Как мы знаем, рекуррентности можно вычислять по аналитической формуле, например, для чисел Фибоначчи получается известная формула Бине: $F_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$. В нашем случае получится похожая формула, содержащая квадратичную иррациональность $\sqrt{2}$.

41.3. Пусть 2 *является квадратичным вычетом по mod p*, например, p = 23, т. е. разрешимо уравнение $x^2 = 2 \bmod p$. **Обоснуйте** алгоритм непосредственного вычисления A по аналитической формуле путем прямого извлечения квадратного корня в конечном поле *mod p*. **Найдите** g_{10000} *mod* 23 этим способом. **Оцените** трудоемкость вычисления g_n *mod p* вашим алгоритмом.

41.4. Пусть теперь 2 не *является квадратичным вычетом по* mod *p,* например, p=19. **Придумайте и обоснуйте** использующий аналитическую формулу алгоритм вычисления чисел $\{g_n\}$ по такому модулю. **Найдите** A этим способом. **Оцените** трудоемкость вычисления g_n *mod p* вашим алгоритмом.

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

Комментарий. Результаты этого простого, но важнейшего упражнения позволяют строить быстрые *вероятностные* алгоритмы, проверяющие простоту чисел. Речь идет о процедурах, которые, получив на вход n-битовую двоичную запись числа, могут быстро[5] 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 повторить такой алгоритм k раз и во всех случаях он выдаст ответ «простое», то с вероятностью $1 - \left(\frac{3}{4}\right)^k$ число действительно будет простым. При таком подходе вероятность $1 - \left(\frac{3}{4}\right)^{1000}$ 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».

---

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

Вход: натуральное число *N*.
Выход: «ДА», если *N* — простое;
«НЕТ», в противном случае.
Случайно выбираем положительное целое число $1<a<N$.
Если НОД(a,N)>1 **То** *Выход:* «НЕТ» **Иначе**
Если $a^{N-1}=1 \ mod \ N$ **То** *Выход*: «ДА» **Иначе** *Выход*: «НЕТ»

Описанный алгоритм — это «почти» полиномиальный вероятностный алгоритм проверки простоты

42.2. Покажите, что «ТЕСТ ФЕРМА» может быть реализован за полиномиальное по входу число операций.

42.3. Пусть известно, что составное число N не является числом Кармайкла[6], т. е. для некоторого натурального a, взаимно-простого с N, выполнено $a^{N-1} \neq 1 \ mod \ N$. Тогда «ТЕСТ ФЕРМА» выдает правильный ответ с вероятностью[7] $\geq \frac{1}{2}$.

43. (Схема RSA). Пусть открытый ключ Боба (11,899). Он хочет послать сообщение (число) за своей подписью. В какую степень он должен его возвести?

44.1. Найдите все решения уравнения $\varphi(n) = 8$.
44.2. Найдите распределение вычетов по показателям в Z_{23}.
44.3. Найдите все первообразные корни *mod* 23.

45. (Хеш-функции). Пусть $h_{a,b}(k)=[(ak+b) \bmod 8] \bmod 5$. Рассмотрим следующее семейство $H=\{h_{a,b} \ a \in Z_9^*, \ b \in Z_9\}$ хеш-функций, отображающих $U = Z_{n_1}$ в Z_{n_2}.
45.1. n_1=?, n_2=?, $|H|$=?
45.2. Докажите или опровергните, что H — универсальное семейство.
45.3. Докажите или опровергните, что H — 2-универсальное семейство.

Д10. Докажите или опровергните, что кодирование в системе RSA: $M \to M^e mod \ N$ биективно отображает отрезок $\{0, \dots, N-1\}$ в себя.

---

[6] 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]**.

[7] Вероятность понимается в «наивном смысле» как отношения числа благоприятных исходов к общему числу исходов.

Задание на 11-ю неделю занятий. Раздел 9 программы.
Дискретное преобразование Фурье. Быстрое преобразование Фурье

46.1. Найдите произведение многочленов $A(x)= 2x^3 + 3x + 1$ и $B(x)=3x^3+x^2+1$, используя рекурсивный O(n log n)-алгоритм БПФ.
46.2. Используя рекурсивный алгоритм БПФ, вычислите символически (*в виде функций от* $\omega= e^{2\pi i/8}$) ДПФ массивов A=(0,0,0,0,2,0,3,1) и B=(0,0,0,0,3,1,0,1).
46.3. Используя предыдущее вычисление, найдите ДПФ массива C в виде функций от ω.
46.4. Используя рекурсивный алгоритм БПФ, найдите коэффициенты многочлена-произведения $C(x)$.
Для этого нужно умножить вектор-столбец, полученный в предыдущем пункте, на обратную матрицу ДПФ. Имеет место тождество:

$$\left(\left|(\omega_n)^{ij}\right|_{i,j=0,1,\dots,n-1}\right)^{-1} = \frac{1}{n}\left(\left|(\omega_n^{-1})^{ij}\right|_{i,j=0,1,\dots,n-1}\right),$$

а потому и при этом вычисление можно использовать БПФ. Считаем, что матрица, и вектор заданы в виде функций (полиномов) от ω, и в этом же формате проводится вычисление. В результате после приведения подобных членов и учета тождеств для корней из единицы должен получиться целочисленный вектор коэффициентов $C(x)$. Проверьте результат.

47. В тексте, выложенном на сайте [8], описан быстрый алгоритм поиска подстрок, использующий БПФ. Текст написан достаточно сжато и вызывает определенные трудности. В этом упражнении нужно обосновать некоторые утверждения статьи. Итак, задача заключается в том, чтобы быстро найти подстроку (образец) $p_0, \dots, p_{m-1}$ в строке (тексте) $t_0, \dots, t_{n-1}$, здесь p_i, t_j– это символы некоторого алфавита. Говорят, что подстрока входит с i-й позиции, если $p_j = t_{i+j},\ j = 0, \dots, m-1$. Если считать буквы алфавита различными целыми числами, то вхождение подстроки с i-й позиции эквивалентно обнулению суммы квадратов: $B_i = \Sigma_{j=0}^{m-1}(p_j - t_{i+j})^2 = \Sigma_{j=0}^{m-1}\left(p_j^2 - 2p_j t_{i+j} + t_{i+j}^2\right) = 0$, а вычисление массива чисел $\{B_i,\ i = 0, \dots, n-m\}$ позволяет определить все места вхождения подстроки в текст. Теперь осталось понять, как провести это вычисление быстро. «Наивный» алгоритм – это тот же прямой перебор и требует $O(mn)$ операций. Но мы же знаем из курса ТРЯП, что есть и более быстрый, линейный, алгоритм. Сейчас мы не будем столь амбициозны и попробуем обосновать $O(n \log m)$-процедуру.
47.1. Покажите, как, используя БПФ, построить $O(n \log n)$-алгоритм вычисления поиска вхождения образца в текст.
В том же тексте показано, как использовать ту же идею для проверки вхождения подстроки, если разрешается использовать символ джокера (в курсе ТРЯП он обозначался

[8] Clifford P., Clifford R. Simple deterministic wildcard matching.//Information Processing Letters. 2007. V.101, P. 53–54.

знаком «?»). Тогда в тех же обозначениях нужно вычислить массив $\{A_i,\ i = 0, \dots, n-m\}$, где $A_i = \Sigma_{j=0}^{m-1}\left(p_j^3 t_{i+j} - 2p_j^2 t_{i+j}^2 + p_j t_{i+j}^3\right)$.

47.2. Покажите, как, используя БПФ, построить $O(n \log n)$-алгоритм вычисления поиска вхождения образца в текст с «джокерами».

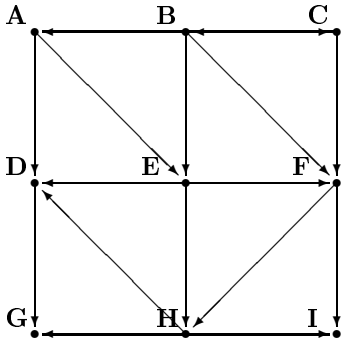
47.3. Покажите, как понизить сложность алгоритмов из **47.1-2** до $O(n \log m)$.

Д11. Проанализируйте зависимость сложности алгоритмов из задачи **47** от величины алфавита. В цитированной оригинальной статье явный ответ на этот вопрос не приводится.

Задание на 12-ю неделю занятий. Раздел 10 программы. Алгоритмы на графах

Поиск в глубину и поиск в ширину

48. Проведите поиск в глубину в графе на рисунке. Используйте алфавитный порядок вершин. Укажите типы всех дуг графа и вычислите для каждой вершины значения функций $d(\cdot)$ и $f(\cdot)$.



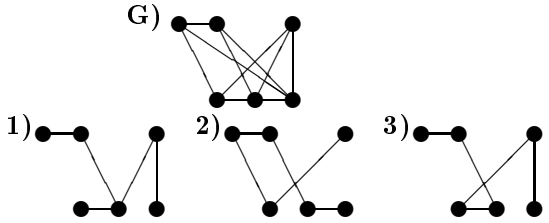
48. Напомним, что **ветвью (branch)** дерева T в неориентированном графе с корнем a называется любое максимальное поддерево T, содержащее a в качестве листа (висячей вершины).

48.1. Докажите или опровергните следующий критерий.

Пусть $G_i,\ i = 1, \dots, k$ – связные компоненты неориентированного графа G. Докажите, что подграф $T \subseteq G$ является графом (лесом) некоторого поиска в глубину, если и только если $T = \cup_{i=1}^{k} T_i(a_i)$, причем каждый подграф $T_i(a_i)$ является таким корневым остовным деревом в G_i с конем a_i, что в G_i ***нет ребер между различными ветвями*** *$T_i(a_i)$.*

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

48.2. В соответствии с полученным в предыдущем пункте критерием установите, какие из нарисованных деревьев являются деревьями поиска в глубину.

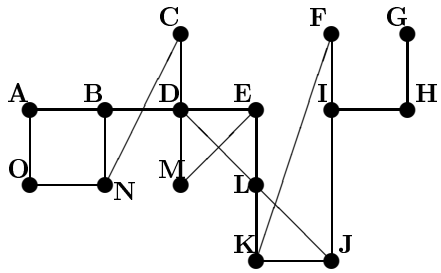


Формат ответа. Пусть, скажем, критерий верен, тогда при положительном ответе нужно указать корень дерева в глубину, а при отрицательном – для каждого возможного выбора корня нужно указать соответствующее ребро между различными ветвями.

Поиск двусвязных компонент графа

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

49.1. Для графа, изображенного на рисунке, укажите точки раздела, мосты и двусвязные компоненты.



49.2. 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.

49.3. Покажите, что G_b – дерево, и постройте соответствующее дерево для G из пункта **49.1.**

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.

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

50.2. 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).*

Д12. Постройте линейный по входу алгоритм, которые, имея на входе граф G и некоторое его остовное дерево T, определяют является ли T деревом поиска-в-глубину при старте с некоторой вершины G.

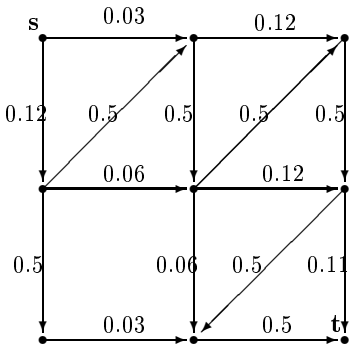
Задание на 13-ю неделю занятий. Раздел 10 программы. Алгоритмы на графах

Кратчайшие пути и связывающие сети

51. Коммуникационная сеть является ориентированным графом, причем 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. В первом приближении считаем, что «вероятности» $r(u,v)$ независимые, и таким образом, надежность передачи сообщения по пути $v_1,\ldots, v_k$ равна $\prod_{i=1}^{k-1} r(v_i, v_{i+1})$.

51.1. Постройте алгоритм нахождения наиболее надежного пути в сети между вершинами s и t и укажите класс сетей, в которых алгоритм будет полиномиальным.

51.2. Проведите вычисления с помощью построенного алгоритма для сети, изображенной на рисунке.



51.3. Постройте наиболее надежную сеть, позволяющую передавать сообщения из вершины *s* в любую другую вершину графа, содержащую *минимальное* число дуг. Под *надежностью* сети понимается произведение надежностей всех входящих в нее дуг.

Д13. 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: $E(G) = \{[(i, j) \to (i+1, j)],\ i = 0,\ldots,m-1,\ j = 0,\ldots,n$ или $[(i, j) \to (i, j + 1)],\ i = 0,\ldots,\ m,\ j = 0,\ldots,\ n-1\}$, причем дугам G приписаны целочисленные веса. Рассмотрим задачу поиска экстремального (самого «тяжелого» или самого «легкого») пути между вершинами (0, 0) и (m, n) (по определению, вес пути равен сумме весов входящих в него ребер). В таком виде – это типичная задача так называемого «динамического программирования», описанная во многих источниках. Для нее легко придумать оптимальный $O(mn)$-алгоритм. (Считаем, что арифметические операции с весами выполняются за $O(1)$.)

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 поиска самого «легкого» пути между вершинами (0, 0) и (m, n) для этого класса грид-графов.

Задание на 14-ю неделю занятий. Раздел 11 программы. Методы решения переборных задач

52. 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:

$$c_1 x_1 + \cdots + c_n x_n = cx \to \max$$
$$a_1 x_1 + \cdots + a_n x_n = ax \le b$$
$$x = (x_1, \ldots, x_n) \in \{0,1\}^n$$

Неформально говоря, мы хотим заполнить рюкзак ограниченного объема наиболее ценными вещами. Если последнее ограничение заменить на $0 \le x_i \le 1$, i=1,..., n, то получится *линейная релаксация дискретной задачи о рюкзаке (ЛРР).* {0,1}-вектор $\tilde{x}$ называется ε-приближенным решением ЗР $0 < \varepsilon \le 1$, если

- $a\ \tilde{x} \le b$;
- $\frac{c\ \tilde{x}}{c\ x_{opt}} \ge \varepsilon$,

здесь x_{opt} — оптимальное решение ЗР.

52.1. Докажите, что ЛРР эффективно решается с помощью «жадного» алгоритма. Алгоритм нужно придумать, доказать его корректность и оценить трудоемкость.

52.2. Покажите, что простая модификация «жадного» алгоритма из предыдущей задачи позволяет найти $\frac{1}{2}$-*приближенное* решение ЗР, а сам «жадный» алгоритм *не гарантирует* получение $\frac{1}{2}$ -приближенного решение ЗР.

Постройте псевдополиномиальные алгоритмы[9] для решения ЗР, имеющие трудоемкость:

52.3. O($n\ b$);

52.4. O($n\ f_{opt}$), где f_{opt} — оптимальное решение ЗР.

Для заданного $\varepsilon > 0$ выберем максимальное натуральное число t из условия $n2^t \le \varepsilon\ f_{opt}$. Отбросим у коэффициентов целевой функции ЗР младшие t разрядов[10]: $c_i \to \widetilde{c_i} = \left\lfloor \frac{c_i}{2^t} \right\rfloor 2^t$. Пусть $\tilde{f}$ — оптимальное решение ЗР с измененной целевой функцией $\widetilde{c_1}\ x_1 + \cdots\ \widetilde{c_n} x_n$.

52.5. Покажите, что $|\tilde{f} - f_{opt}| \le \varepsilon\ f_{opt}$.

52.6. Используя пункты **53.2**, **53.4** и **53.5**, постройте алгоритм, находящий ε-приближенное решение ЗР. Алгоритм должен выполнять $O(\frac{n^2}{\varepsilon})$ операций над O($max\ \{b, \frac{n}{\varepsilon}\}$) числами, ***используя память*** $O(\frac{n}{\varepsilon})$.

52.7. Используя построенный в **52.6** алгоритм, найдите ε-оптимальное решение следующей задачи о рюкзаке.

$\varepsilon = 0.25$

$58x_1 + 70x_2 + 59x_3 + 8x_4 + 60x_5 + 240x_6 + 250x_7 + 249x_8 \to max$

$33x_1 + 5x_2 + 34x_3 + 5x_4 + 31x_5 + 185x_6 + 190x_7 + 191x_8 \le 256$

$x_i \in \{0,1\}$, i=1,2,…,8.

---

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

[10] Это преобразование называется шкалированием (scaling).

Задание на 15-ю неделю занятий. Раздел 15 программы. Вероятностные алгоритмы

53. Язык 2-ВЫПОЛНИМОСТЬ состоит из выполнимых КНФ, в которых каждый дизъюнкт содержит не более двух литералов. В следующей задаче будут изложены два эффективных алгоритма проверки выполнимости формул из языка 2-ВЫПОЛНИМОСТЬ, т. е. будет доказано, что 2-ВЫПОЛНИМОСТЬ $\in P$. Пусть 2-КНФ имеет n литералов и m дизъюнктов.

53.1. Сведение к задаче на графах. Построим по 2-КНФ ориентированный граф G с вершинами, помеченными литералами и их отрицаниями. Каждой дизъюнкции $x \vee y$ в 2-КНФ отвечает пара ориентированных дуг $\neg x \to y$ и $\neg y \to x$ в G. Неформально дуги отвечают импликациям согласно формуле $x \vee y \Leftrightarrow \neg x \to y$, так что путям в графе отвечает последовательность присваиваний истинностных значений переменным.

Докажите или опровергните, что 2-КНФ невыполнима тогда и только тогда, когда в графе G никакая пара вершин $(x, \neg x)$ не лежит на одном контуре. Если в указанном виде критерий неверен, то модифицируйте его до корректного и постройте полиномиальный алгоритм для проверки выполнимости 2-КНФ.

53.2. Случайный поиск[11]. Сначала всем переменным присваивается значение TRUE. На каждой итерации, пока формула невыполнима, берется 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 ½ на 1. Таким образом, наш алгоритм можно интерпретировать как случайное блуждание на отрезке $[0,1,\ldots,n]$.

Покажите, что для выполнимой 2-КНФ среднее число итераций алгоритма (математическое ожидание числа итераций) равно $O(n^2)$.

54. Задача о минимальном разрезе в неориентированном графе заключается в том, чтобы разбить вершины графа на два дизъюнктных подмножества $(S, \bar{S}), S \neq V, S \neq \emptyset$ так, чтобы минимизировать число ребер с концами в разных долях. Конечно, для ее решения можно применить потоковый алгоритм, но мы рассмотрим простую вероятностную процедуру. При этом основной будет операция стягивания ребра (кратные ребра остаются, а петли удаляются). Граф, полученный стягиванием ребра $(x, y) \in E$, обозначим $G/_{xy}$. Первоначальная идея заключается в следующем: при стягивании ребер величина минимального разреза *не убывает* (пока в графе остается не менее двух вершин), так что если стянуть все ребра $\{e_1, \ldots, e_p\}$, *не входящие в минимальный разрез*, то останется пара

[11] Начиная с этой задачи, полезно ознакомиться с **[Кормен 1,** §§ **6.2—6.3]** (соответственно, **[Кормен 2, §§ С.2—С.3]**).

вершин, соединенная k ребрами, где k — величина минимального разреза в G. Остается понять, как часто реализуется подобная ситуация, если ребра стягиваются случайно.

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

$G_0 \leftarrow G;\ i \leftarrow 0;$

Пока $|V(G_i)|>2$ **Выполнить**

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

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

На выходе из цикла получаем (мульти)граф $\widetilde{G}$, имеющий две вершины.

Выход [разрез в исходном графе G, отвечающий разрезу $\widetilde{G}$].

Время работы алгоритма $O(n^2)$.

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

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

Если привлечь дополнительные соображения, то можно понизить трудоемкость до $O(n^2\ log^{O(1)}\ n)$. При этом алгоритм столь же прост и допускает параллелизацию.

Финальный тест. Темы: весь материал курса.

Литература

1. **[АХУ]** *Ахо А., Хопкрофт Д., Ульман Д.* Построение и анализ вычислительных алгоритмов. М.: Мир, 1979.
2. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982. Электронный вариант: http://trpl.narod.ru/NPC-book.htm
3. **[Кормен 1]** *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы. Построение и анализ. М.: МЦНМО, 2002.
4. **[Кормен 2]** *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы. Построение и анализ. Изд. 2-е. М.: Вильямс, 2005.
5. *Кузюрин Н., Фомин С.* Эффективные алгоритмы и сложность вычислений. М.: МФТИ, 2007.

Дополнительная литература

6. *Верещагин Н., Шень А.* Вычислимые функции. М.: МЦНМО, 1999. Электронный вариант: www.mccme.ru/free-books
7. *Виноградов И.* Основы теории чисел. М.-Л.: Гостехиздат, 1952.
8. *Вялый М., Журавлев Ю., Флеров Ю.* Дискретный анализ. Основы высшей алгебры. М.: МЗ Пресс, 2007.
9. **[К-Ш-В]** *Китаев А., Шень А., Вялый М.* Классические и квантовые вычисления. М.: МНЦМО-ЧеРо, 1999
10. **[Хинчин]** *Хинчин А.* Цепные дроби. М.: Наука, 1979.
11. *Шень А.* Программирование. Теоремы и задачи. М.: МЦНМО, 2007. Электронный вариант: www.mccme.ru/free-books

12. *Lovasz L.* Computational Complexity. Электронный вариант www.cs.elte.hu/~lovasz/complexity.pdf

13. *Кнут Д.* Искусство программирования для ЭВМ. Т. 1. М.: Мир, 1976; т. 2. М.: Мир, 1977; т. 3. М.: Мир, 1978; т. 1. М.: Вильямс, 2006; т. 2. М.: Вильямс, 2007; т. 3. М.: Вильямс, 2007; т. 4. М.: Вильямс, 2013.