Содержание

Вступление

Однажды (это случилось 11 ноября 2003 года) я получил письмо по электронной почте, автор которого писал мне:

> «Я должен сказать, что на сайте университета Вальядолида Вы положили начало новой ЦИВИЛИЗАЦИИ. Своими книгами (он имел в виду «Задачи по программированию: пособие по подготовке к олимпиадам по программированию» [60], написанное в соавторстве со Стивеном Скиеной) Вы вдохновляете людей на подвиги. Желаю Вам долгих лет жизни, чтобы служить человечеству, создавая новых супергероев – программистов».

Хотя это было явным преувеличением, письмо заставило меня задуматься. У меня была мечта: создать сообщество вокруг проекта, который я начал как часть моей преподавательской работы в университете Вальядолида. Я мечтал создать сообщество, которое объединило бы множество людей с разных концов света, связанных единой высокой целью. Выполнив несколько запросов в поисковике, я быстро нашел целое онлайн-сообщество, объединявшее несколько сайтов, обладавших всем тем, чего не хватало сайту университета Вальядолида.

Сайт «Методы решения задач» Стивена Халима, молодого студента из Индонезии, показался мне одним из наиболее интересных. Я увидел, что однажды мечта может стать реальностью, потому что на этом сайте я нашел результат кропотливой работы гения в области алгоритмов и программирования. Более того, цели, о которых он говорил, вполне соответствовали моей мечте: служить человечеству. И еще я узнал, что у него есть брат, Феликс Халим, разделяющий его увлечение программированием.

До начала нашего сотрудничества прошло довольно много времени, но, к счастью, все мы продолжали параллельно работать над реализацией этой мечты. Книга, которую вы сейчас держите в руках, является лучшим тому доказательством.

Я не могу представить лучшего дополнения для «Онлайн-арбитра» университета Вальядолида. Эта книга использует множество примеров с сайта университета Вальядолида, тщательно отобранных и разбитых по категориям по типам задач и методам их решения. Она оказывает огромную помощь пользователям данного сайта. Разобрав и решив большинство задач по программированию из этой книги, читатель сможет легко решить не менее 500 задач, предложенных «Онлайн-арбитром» университета Вальядолида, и попасть в число 400–500 лучших среди 100 000 пользователей «Онлайн-арбитра».

Книга «Олимпиады по программированию: повышая уровень соревнований по программированию» также подходит для программистов, которые хотят улучшить свои позиции на региональных и международных соревнованиях по программированию. Ее авторы прошли через эти соревнования (ICPC и IOI) вначале как участники, а теперь и как тренеры. Она также рекомендуется новичкам – как говорят Стивен и Феликс во введении: «Книгу рекомендуется прочесть не один, а несколько раз».

Кроме того, она содержит исходный код на C++, реализующий описанные алгоритмы. Понимание задачи – это одно, знание алгоритма ее решения – другое, а правильная реализация решения через написание краткого и эффективного кода – третья непростая задача. Прочитав эту книгу трижды, вы поймете, что ваши навыки программирования значительно улучшились и, что еще важнее, вы стали более счастливым человеком, чем были раньше.

Мигель А. Ревилла,
Университет Вальядолида,
создатель сайта «Онлайн-арбитр»,
член Международного оргкомитета ACM ICPC
http://uva.onlinejudge.org; http://livearchive.onlinejudge.org

Участники финального мирового турнира в Варшаве, 2012.
Слева направо: Фредерик Нимеля, Карлос, Мигель Ревилла, Мигель-младший, Феликс, Стивен

Предисловие

Эта книга обязательна к прочтению для каждого программиста, участвующего в соревнованиях по программированию. Овладеть содержанием данной книги необходимо (но, возможно, недостаточно) для того, чтобы сделать шаг вперед от простого обычного программиста до одного из лучших программистов в мире.

Для кого написана эта книга:

- для студентов университетов, участвующих в ежегодных всемирных студенческих соревнованиях по программированию ACM (ICPC [66]) в качестве участников региональных соревнований (включая финальные мировые турниры);
- для учащихся средних или старших классов школ, принимающих участие в ежегодной Международной олимпиаде по информатике (IOI) [34] (включая национальные или региональные олимпиады);
- для тренеров сборных команд по программированию, преподавателей и наставников, которые ищут полноценные учебные материалы для своих воспитанников [24];
- для всех, кто любит решать задачи по программированию. Для тех, кто больше не имеет права участвовать в ICPC, проводятся многочисленные состязания по программированию, в том числе TopCoder Open, Google CodeJam, интернет-конкурс по решению задач (IPSC) и т. д.

Требования к уровню подготовки

Эта книга *не* предназначена для начинающих программистов. Она написана для читателей, которые имеют хотя бы элементарные знания из области методологии программирования, знакомы хотя бы с одним из двух языков программирования – C/C++ или Java (а еще лучше с обоими), освоили начальный курс по структурам данных и алгоритмам (обычно он включается в программу первого года обучения в университете для специальностей в области информатики) и знакомы с простым алгоритмическим анализом (по крайней мере, им знакома нотация Big O). Третье издание было значительно переработано и дополнено, так что эта книга также может быть использована в качестве *дополнительного материала для начального курса по структурам данных и алгоритмам*.

Участникам ACM ICPC

Мы знаем, что вряд ли возможно одержать победу в ACM ICPC, просто прочитав новую версию этой книги и усвоив ее содержание. Хотя мы включили в нее много полезного материала – гораздо больше, чем содержалось в пер-

вых двух изданиях, – мы понимаем, что для достижения этой цели требуется гораздо больше, чем может дать данная книга. Для читателей, которые хотят большего, мы приводим дополнительные ссылки на полезные источники. Однако мы полагаем, что после освоения содержания этой книги ваша команда будет чувствовать себя намного увереннее в будущих состязаниях ICPC. Мы надеемся, что эта книга послужит как источником вдохновения, так и стимулом к участию в соревнованиях ACM ICPC во время вашей учебы в университете.

Участникам IOI

Большая часть наших советов для участников ACM ICPC пригодится и вам. Программы ACM ICPC и IOI в значительной мере схожи, однако IOI на данный момент исключает темы, перечисленные в табл. П1. Вы можете пропустить соответствующие главы этой книги, отложив их до тех пор, пока не поступите в университет (и не присоединитесь к команде этого университета, участвующей в ACM ICPC). Тем не менее полезно изучить эти методы заранее, так как дополнительные знания могут помочь вам в решении некоторых задач в IOI.

Мы знаем, что нельзя получить медаль IOI, просто основательно проштудировав эту книгу. Мы включили многие разделы программы IOI в данную книгу и надеемся, что вы сможете добиться достойного результата на олимпиадах IOI. Однако мы хорошо понимаем, что задачи, предлагаемые на IOI, требуют сильных навыков решения задач и огромного творческого потенциала – тех качеств, которые вы не сможете получить, только читая учебник. Книга может дать вам знания, но в конечном итоге вы должны проделать огромную работу. С практикой приходит опыт, а с опытом приходит навык. Так что продолжайте практиковаться!

Слева направо: Дэниел, мистер Чонг, Раймонд, Стивен, Чжан Сюн, д-р Рональд, Чуанци

Таблица П1. Темы, не входящие в программу IOI [20]

Тема	**В этой книге:**
Структуры данных: система непересекающихся множеств	Раздел 2.4.2
Теория графов: нахождение компонентов сильной связности, поток в сети, двудольные графы	Разделы 4.2.1, 4.6.3, 4.7.4
Математические задачи: операции с очень большими числами BigInteger, теория вероятностей, игра «Ним»	Разделы 5.3, 5.6, 5.8
Обработка строк: суффиксные деревья и массивы	Раздел 6.6
Более сложные темы: A*/IDA*	Раздел 8.2
Нестандартные задачи	Глава 9

Преподавателям и наставникам

Эта книга используется как учебное пособие на курсе Стивена CS3233 «Олимпиадное программирование» в Школе программирования Национального университета Сингапура. Продолжительность курса CS3233 составляет 13 учебных недель, его примерная программа приведена в табл. П2. Слайды для этого курса в формате PDF опубликованы на веб-сайте данной книги. Коллегам – преподавателям и наставникам рекомендуется изменять программу в соответствии с потребностями студентов. В конце каждой главы данной книги приводятся подсказки или краткие решения заданий, не помеченных звездочкой. Некоторые из помеченных заданий довольно сложны и не имеют ни ответов, ни решений. Их можно использовать в качестве экзаменационных вопросов или конкурсных заданий (разумеется, сначала нужно их решить).

Эта книга также применяется в качестве дополнительного материала в программе курса CS2010 «Структуры данных и алгоритмы», в основном для иллюстрации реализации нескольких алгоритмов и выполнения заданий по программированию.

Таблица П2. Программа курса CS3233 «Олимпиадное программирование»

Неделя	Тема	В этой книге:
01	Введение	Глава 1, разделы 2.2, 5.2, 6.2–6.3, 7.2
02	Структуры данных и библиотеки	Глава 2
03	Полный поиск, стратегия «разделяй и властвуй», «жадный» алгоритм	Разделы 3.2–3.4; 8.2
04	Динамическое программирование: основы	Разделы 3.5; 4.7.1
05	Динамическое программирование: техники	Разделы 5.4; 5.6; 6.5; 8.3
06	Командные соревнования в середине семестра	Главы 1–4; часть главы 9
–	Перерыв в середине семестра (домашнее задание)	
07	Графы, часть 1 (поток в сети)	Раздел 4.6; часть главы 9
08	Графы, часть 2 (поиск соответствий)	Раздел 4.7.4; часть главы 9
09	Математика (обзор)	Глава 5
10	Обработка строк (основные навыки, массив суффиксов)	Глава 6
11	(Вычислительная) Геометрия (библиотеки)	Глава 7
12	Более сложные темы	Раздел 8.4; часть главы 9
13	Заключительные командные соревнования	Главы 1–9 (не ограничиваясь только ими)

Для курсов по структурам данных и алгоритмам

В этом издании содержание данной книги было переработано и дополнено таким образом, что первые четыре главы книги стали более доступными для студентов первого курса, специализирующихся в области информатики и теории вычислительных систем. Темы и упражнения, которые мы сочли относительно сложными для начинающих, были перенесены в главы 8 и 9. Надеемся, что студенты, только начинающие свой путь в области компьютерных наук, не испугаются трудностей, просматривая первые четыре главы.

Глава 2 была основательно переработана. Ранее раздел 2.2 представлял собой просто список классических структур данных и их библиотек. В данном издании мы расширили описание и добавили множество упражнений, чтобы эту книгу можно было также использовать как пособие для курса по структурам данных, особенно с точки зрения *деталей реализации*.

О четырех подходах к решению задач, обсуждаемых в главе 3 этой книги, часто рассказывается в различных курсах по *алгоритмам*.

Текст предисловия также был переработан, чтобы помочь новым студентам в области информатики ориентироваться в структуре книги.

Часть материала из главы 4 можно использовать в качестве дополнительной литературы либо в качестве наглядного примера реализации. Этот материал пригодится для повышения уровня знаний в области *дискретной математики* [57, 15] или для начального курса по *алгоритмам*. Мы также представили новый взгляд на методы динамического программирования как на алгоритмы, использующие ориентированные ациклические графы. К сожалению, во многих учебниках по компьютерным наукам такие темы до сих пор встречаются редко.

Всем читателям

Поскольку эта книга охватывает множество тем, обсуждаемых вначале более поверхностно, затем более глубоко, ее рекомендуется прочитать не один, а несколько раз. Книга содержит много упражнений (их около 238) и задач по программированию (их около 1675); практически в каждом разделе предлагаются различные задания и упражнения. Вы можете сначала пропустить эти упражнения, если решение слишком сложно или требует дополнительных знаний и навыков, чтобы потом вернуться к ним после изучения следующих глав. Решение предложенных задач углубит понимание понятий, изложенных в этой книге, поскольку они обычно включают интересные практические аспекты обсуждаемой темы. Постарайтесь их решить – время, потраченное на это, точно не будет потрачено впустую.

Мы считаем, что данная книга актуальна и будет интересна многим студентам университетов и старших классов. Олимпиады по программированию, такие как ICPC и IOI, по крайней мере, еще много лет будут иметь похожую программу. Новые поколения студентов должны стремиться понять и усвоить элементарные знания из этой книги, прежде чем переходить к более серьезным задачам. Однако слово «элементарные» может вводить в заблуждение – пожалуйста, загляните в содержание, чтобы понять, что мы подразумеваем под «элементарным».

Как ясно из названия этой книги, ее цель – развить навыки программирования и тем самым поднять уровень всемирных олимпиад по программированию, таких как ICPC и IOI. Поскольку все больше участников осваивают ее содержание, мы надеемся, что 2010 год (год, когда было опубликовано первое издание этой книги) стал переломным моментом, знаменующим существенное повышение стандартов соревнований по программированию. Мы надеемся помочь большему количеству команд справиться более чем с двумя ($\geqslant 2$) задачами на будущих олимпиадах ICPC и позволить большему количеству участников получить более высокие ($\geqslant 200$) баллы на будущих олимпиадах IOI. Мы также надеемся, что многие тренеры ICPC и IOI во всем мире (особенно в Юго-Восточной Азии) выберут эту книгу и оценят помощь, которую она оказывает при выборе материала для подготовки к соревнованиям тем, без кого студенты не могут обойтись на олимпиадах по программированию, – преподавателям и наставникам. Мы, ее авторы, будем очень рады, что нам удалось внести свой вклад в распространение необходимых «элементарных» знаний для олимпиадного программирования, поскольку в этом заключается основная цель нашей книги.

Принятые обозначения

Эта книга содержит множество примеров кода на языке C/C++, а также отдельные примеры кода на Java (в разделе 5.3). Все фрагменты кода, включенные в книгу, выделяются моноширинным шрифтом.

Для кода на C/C++ в данной книге мы часто используем директивы `typedef` и макросы – функции, которые обычно применяются программистами, участ-

вующими в соревнованиях, для удобства, краткости и скорости кодирования. Однако мы не можем использовать аналогичные методы для Java, поскольку они не содержат аналогичных функций. Вот несколько примеров наших сокращений для кода C/C++:

```
// Отключение некоторых предупреждений компилятора (только для пользователей VC++)
#define _CRT_SECURE_NO_DEPRECATE

// Сокращения, используемые для наиболее распространенных типов данных в олимпиадном
программировании
typedef long long       ll;                    // комментарии, которые смешиваются с кодом,
typedef pair<int, int>  ii;                              // выравниваются подобным образом
typedef vector<ii>      vii;
typedef vector<int>     vi;
#define INF 1000000000              // 1 миллиард, что предпочтительнее, чем сокращение 2B,
                                                        // в алгоритме Флойда-Уоршелла
// Общие параметры memset
//memset(memo, -1, sizeof memo);            // инициализация таблицы, сохраняющей значения
вычислений
                                        // в динамическом программировании, значением -1
//memset(arr, 0, sizeof arr);                              // очистка массива целых чисел
// Мы отказались от использования "REP" and "TRvii" во втором и последующих
// изданиях, чтобы не запутывать программистов
```

Следующие сокращения часто используются как в примерах кода на C/C++, так и в примерах кода на Java:

```
// ans = a ? b : c;                         // для упрощения: if (a) ans = b; else ans = c;
// ans += val;                             // для упрощения: ans = ans + val; и ее вариантов
// index = (index + 1) % n;                          // index++; if (index >= n) index = 0;
// index = (index + n - 1) % n;                // index--; if (index < 0) index = n - 1;
// int ans = (int)((double)d + 0.5);                 // для округления до ближайшего целого
// ans = min(ans, new_computation);                                  // сокращение min/max
// альтернативная форма записи, не используемая в этой книге: ans <?= new_computation;
// в некоторых примерах кода используются обозначения: && (AND) и || (OR)
```

Категории задач

К маю 2013 года Стивен и Феликс совместно решили 1903 задачи по программированию, размещенные на сайте университета Вальядолида (что составляет 46,45 % от общего числа задач, размещенных на этом сайте). Около 1675 из них мы разбили по категориям и включили в эту книгу. В конце 2011 года список задач «Онлайн-арбитра» на сайте университета Вальядолида пополнился некоторыми задачами из Live Archive. В этой книге мы приводим оба индекса задач, однако основным идентификатором, используемым в разделе предметного указателя данной книги, является номер задачи, присвоенный ей на сайте университета Вальядолида. Задачи распределялись по категориям в соответствии со схемой «балансировки нагрузки»: если задачу можно разделить на две или более категорий, то она будет отнесена к категории, к которой в настоящий момент относится меньшее число задач. Таким образом, вы можете

заметить, что некоторые задачи были «ошибочно» классифицированы, и категория, к которой отнесена данная задача, может не соответствовать методу, который вы использовали для ее решения. Мы можем лишь гарантировать, что если задача X определена в категорию Y, то нам удалось решить задачу X с помощью метода, обсуждаемого в разделе, посвященном категории задач Y.

Мы также ограничили число задач в каждой категории: в каждую из категорий мы включили не более 25 (двадцати пяти) задач, разбив их на отдельные дополнительные категории.

Если вы хотите воспользоваться подсказками к любой из решенных задач, воспользуйтесь указателем в конце этой книги, а не пролистывайте каждую главу – это позволит сэкономить время. Указатель содержит список задач с сайта университета Вальядолида / Live Archive с номером задачи (используйте двоичный поиск!) и номера страниц, на которых обсуждаются упомянутые задачи (а также структуры данных и/или алгоритмы, необходимые для их решения). В третьем издании подсказки занимают более одной строки, и они стали гораздо понятнее, чем в предыдущих изданиях.

Используйте различные категории задач для тренировки навыков в различных областях! Решение, по крайней мере, нескольких задач из каждой категории (особенно тех, которые мы пометили **знаком** *) – это отличный способ расширить ваши навыки решения задач. Для краткости мы ограничились максимум тремя основными задачами в каждой категории, пометив их особо.

Изменения во втором издании

Первое и второе издания этой книги существенно различаются. Авторы узнали много нового и решили сотни задач из области программирования за один год, разделяющий эти два издания. Мы получили отзывы от читателей, в частности от студентов Стивена, прослушавших курс CS233 во втором семестре 2010/2011 года, и включили их предложения во второе издание.

Краткий перечень наиболее важных изменений во втором издании:

- во-первых, изменился дизайн книги. Увеличилась плотность информации на каждой странице. В частности, мы уменьшили междустрочный интервал, сделали более компактным размещение мелких рисунков на страницах. Это позволило нам не слишком сильно увеличивать толщину книги, существенно расширив ее содержание;
- были исправлены некоторые незначительные ошибки в наших примерах кода (как те, что приведены в книге, так и их копии, опубликованные на ее веб-сайте). Все примеры кода теперь имеют более понятные комментарии;
- исправлены некоторые опечатки, грамматические и стилистические ошибки;
- помимо того что мы улучшили описание многих структур данных, алгоритмов и задач программирования, мы также существенно расширили содержание каждой из глав:
  1) добавлено множество новых специальных задач в начале книги (раздел 1.4);

2) в книгу включено обсуждение булевых операций (операций над битами) (раздел 2.2), неявных графов (раздел 2.4.1) и структур данных дерева Фенвика (раздел 2.4.4);
3) расширена часть, посвященная динамическому программированию: дано более четкое объяснение динамического программирования по принципу «от простого к сложному», приведено решение $O(n \log k)$ для задачи LIS, решение о нахождении суммы подмножества в задаче о рюкзаке (Рюкзак 0–1), рассмотрено решение задачи о коммивояжере методами динамического программирования (с использованием операций с битовыми масками) (раздел 3.5.2);
4) реорганизован материал по теории графов в обсуждении следующих тем: методы обхода графа (поиск в глубину и поиск в ширину), построение минимального остовного дерева, нахождение кратчайших путей (из заданной вершины во все вершины и между всеми парами вершин), вычисление максимального потока и обсуждение задач, относящихся к специальным графам. Добавлены новые темы: обсуждение алгоритма Прима, использование методов динамического программирования на графах при обходе неявных направленных ациклических графов (раздел 4.7.1), эйлеровы графы (раздел 4.7.3) и алгоритм поиска аугментальных цепей (раздел 4.7.4);
5) переработан материал, касающийся обсуждения математических методов (глава 5): специальные задачи, использование BigInteger (в Java), комбинаторика, теория чисел, теория вероятностей, поиск циклов, теория игр (новый раздел) и степень (квадратной) матрицы (новый раздел). Улучшена подача материала с точки зрения легкости восприятия их читателями;
6) добавлен материал, позволяющий получить элементарные навыки обработки строк (раздел 6.2), расширен круг задач, связанных с работой со строками (раздел 6.3), включая сопоставление строк (раздел 6.4), расширен раздел, посвященный теме суффиксных деревьев / массивов (раздел 6.6);
7) расширен набор геометрических библиотек (глава 7), в частности библиотек, позволяющих выполнять операции над точками, линиями и многоугольниками;
8) добавлена глава 8, в которой обсуждаются вопросы декомпозиции задач, расширенные методы поиска (A*, поиск с ограничением глубины, итеративное углубление, IDA*), расширенные методы динамического программирования (использование битовой маски, задача о китайском почтальоне, обзор общих положений динамического программирования, обсуждение лучших применений метода динамического программирования и некоторые более сложные темы);

- был переработан и улучшен иллюстративный материал книги. Также были добавлены новые иллюстрации, помогающие сделать объяснение материала более понятным;
- первое издание книги было написано в основном с точки зрения участника ICPC и программиста на C++. Материал второго издания более

сбалансирован и включает в себя информацию, чья целевая аудитория – участники соревнований IOI. Также во втором издании намного увеличилось число примеров на Java. Однако мы пока не включаем в книгу примеры на других языках программирования;

- это издание книги также включает материалы веб-сайта Стивена «Методы решения»: «однострочные подсказки» для каждой задачи и список задач с указанием категорий в конце этой книги. Теперь решение 1000 задач из списка задач «Онлайн-арбитра» на сайте университета Вальядолида – вполне достижимая цель (мы считаем, что это по силам серьезному студенту 4-го курса университета);
- некоторые примеры в первом издании используют устаревшие задачи. Во втором издании эти примеры были заменены или обновлены;
- Стивен и Феликс решили добавить в эту книгу еще 600 задач по программированию из «Онлайн-арбитра» и онлайн-архива с сайта университета Вальядолида. Мы также добавили в книгу множество практических упражнений с подсказками или краткими решениями;
- книга пополнилась краткими биографиями изобретателей структур данных и авторов алгоритмов, заимствованными из Википедии [71] и из других источников, чтобы вы могли немного больше узнать о людях, оставивших свой след в области программирования.

Изменения в третьем издании

За два года, прошедших с момента выхода второго издания, мы значительно улучшили и дополнили материалы книги, включенные в третье издание.

Краткий перечень наиболее важных изменений в третьем издании:

- мы немного увеличили размер шрифта в третьем издании (12 пунктов) по сравнению со вторым изданием (11 пунктов) и изменили размеры полей. Надеемся, что многие читатели отметят, что текст стал более читабельным. Мы также увеличили размер иллюстраций. Это, однако, привело к тому, что книга стала значительно толще;
- мы изменили макет книги. Теперь почти каждый раздел начинается с новой страницы;
- мы добавили еще *много* практических упражнений в каждый из разделов книги – как не отмеченных звездочкой (предназначенных для самопроверки; подсказки и решения для этих задач приведены в конце каждой главы), так и помеченных звездочкой * (дополнительных задач; решение не предусмотрено). Материалы практических упражнений соседствуют с обсуждением соответствующей темы в тексте глав;
- Стивен и Феликс решили еще 477 задач по программированию из «Онлайн-арбитра» и онлайн-архива с сайта университета Вальядолида (UVa), которые впоследствии были добавлены в эту книгу. Таким образом, мы сохранили значительный (около 50 %, а точнее 46,45 %) охват задач, представленных на сайте «Онлайн-арбитра», даже несмотря на то, что «Онлайн-арбитр» сильно вырос за тот же период времени. Эти новые

задачи были выделены курсивом. Некоторые из новых задач заменили старые, которые рекомендуется обязательно решить. Все задачи теперь всегда размещаются в конце раздела;

- сейчас мы с уверенностью можем сказать, что способные студенты могут достичь впечатляющих успехов в решении задач (более 500 решенных задач «Онлайн-арбитра» на сайте университета Вальядолида) всего за один университетский семестр (4 месяца), прочитав эту книгу;
- перечень новых (или переработанных) материалов по главам:
  1) введение главы 1 было адаптировано для читателей, которые плохо знакомы с олимпиадным программированием. Мы разработали более строгие форматы ввода-вывода (I/O) для типичных задач программирования и общих процедур их решения;
  2) мы добавили еще одну линейную структуру данных 'deque' в разделе 2.2, а также углубили разъяснение практически всех структур данных, обсуждаемых в главе 2, особенно в разделах 2.3 и 2.4;
  3) в главе 3 мы более подробно обсудим различные методы полного перебора: вложенные циклы, итеративную генерацию подмножеств/перестановок и рекурсивный поиск. Новое: интересный трюк для написания и печати решений с использованием «нисходящего» метода динамического программирования, обсуждение алгоритма Кадана о поиске максимальной суммы диапазона для одномерного случая;
  4) в главе 4 мы заменили белые/серые/черные метки (унаследованные из источника [7]) их стандартными обозначениями, заменили термин «максимальный поток» на «сетевой поток». Мы также сослались на опубликованную научную работу автора алгоритма, чтобы читатель мог ознакомиться и понять оригинальную идею, лежащую в основе алгоритма. Мы обновили диаграммы неявного направленного ациклического графа в классических задачах динамического программирования в разделе 3.5;
  5) в главе 5 мы расширили круг обсуждаемых специальных задач математики, обсуждение интересной операции Java BigInteger: `isProbablePrime`, добавили и расширили несколько часто используемых формул комбинаторики и разновидностей алгоритмов решета, дополнили и пересмотрели разделы по теории вероятностей (раздел 5.6), поиску циклов (раздел 5.7) и теории игр (раздел 5.8);
  6) в главе 6 мы переписали раздел 6.6, сделав понятнее объяснение суффиксного бора / дерева / массива суффиксов, и заново определили понятие завершающего символа;
  7) главу 7 мы разбили на два основных раздела и улучшили качество кода библиотеки;
  8) в главу 8 (а также последующую главу 9) включены наиболее сложные темы, ранее обсуждавшиеся в главах 1–7 второго издания. Новое: обсуждение более сложной процедуры возвратной рекурсии, поиска в пространстве состояний, метода «встреча посередине», некоторые неочевидные приемы с использованием сбалансированного дерева двоичного поиска в качестве мемотаблицы и углубленный раздел о декомпозиции задач;

9) новая глава 9. Добавлены различные редкие темы, появляющиеся время от времени на олимпиадах по программированию. Некоторые из них просты, однако многие достаточно сложны и могут сильно повлиять на ваш рейтинг в соревнованиях по программированию.

Сайты, сопутствующие книге

Официальный веб-сайт этой книги имеет адрес sites.google.com/site/stevenhalim, на нем размещена электронная версия примеров исходного кода из данной книги и слайды в формате PDF, используемые в курсе Стивена CS3233.

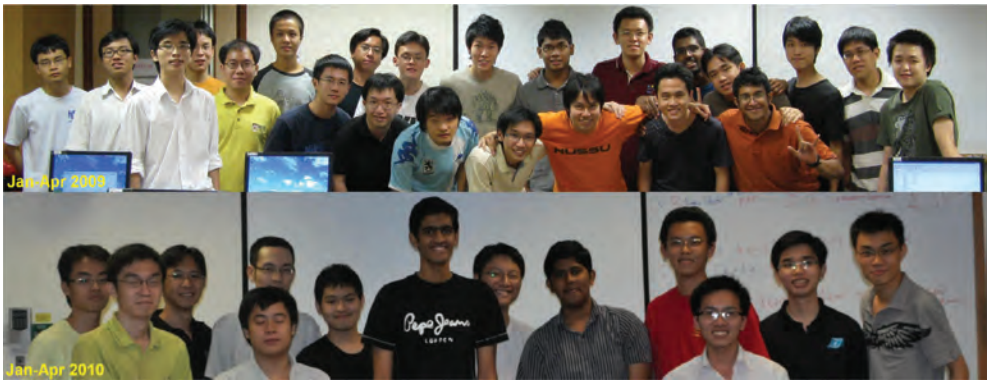
Все задачи по программированию здесь собраны в приложении uhunt.felixhalim.net и размещены в «Онлайн-арбитре» сайта университета Вальядолида: uva.onlinejudge.org.

Новое в третьем издании: многие алгоритмы теперь сопровождаются интерактивной визуализацией: www.comp.nus.edu.sg/~stevenha/visualization.

Благодарности к первому изданию

От Стивена: я хочу поблагодарить:

- Бога, Иисуса Христа и Святого Духа за данный мне талант и страсть к олимпиадному программированию;
- мою любимую жену Грейс Сурьяни за то, что она позволила мне потратить драгоценное время, отведенное нам с ней, на этот проект;
- моего младшего брата и соавтора, Феликса Халима, за то, что он поделился многими структурами данных, алгоритмами и приемами программирования, внеся значительный вклад в улучшение этой книги;
- моего отца Линь Цзе Фонга и мать Тан Хой Лан за данное нам воспитание и мотивацию, необходимые для достижения хороших результатов в учебе и труде;



- Школу программирования Национального университета Сингапура за предоставленную мне возможность работать в этом университете и пре-

подавать учебный курс CS3233 «Олимпиадное программирование», на основе которого и была создана эта книга;
- профессоров и преподавателей, работавших и работающих в Национальном университете Сингапура, давших мне навыки олимпиадного программирования и наставничества: профессора Эндрю Лима Леонга Чи, доцента Тана Сун-Тека, Аарона Тан Так Чой, доцента Сунг Вин Кина, доктора Алана Ченга Хо-Луна;
- моего друга Ильхама Вината Курния за корректуру рукописи первого издания;
- помощников преподавателей курса CS3233 и наставников команд – участников олимпиад ACM ICPC в NUS: Су Чжана, Нго Минь Дюка, Мелвина Чжан Чжиюна, Брамандию Рамадхана;
- моих студентов, обучавшихся на курсе CS3233 во втором семестре 2008/2009 учебного года и вдохновивших меня на создание заметок к моим занятиям, а также студентов, посещавших этот курс во втором семестре 2009/2010 учебного года, которые проверили содержание первого издания этой книги и положили основу онлайн-архива задач Live Archive.

Благодарности ко второму изданию

От Стивена: я также хочу поблагодарить:

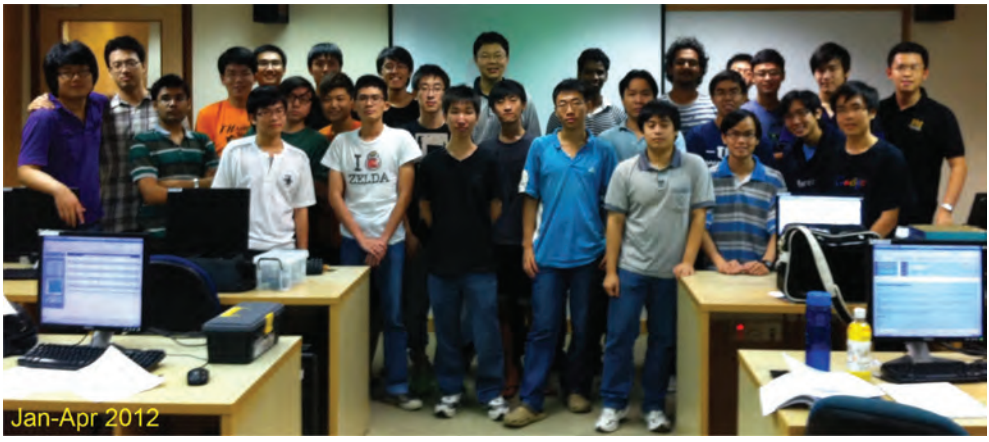
- первых 550 покупателей первого издания (по состоянию на 1 августа 2011 года). Ваши положительные отзывы вдохновляют нас!
- помощника преподавателя курса CS3233 Национального университета Сингапура Виктора Ло Бо Хуай;
- моих студентов, обучавшихся на курсе CS3233 во втором семестре 2010/2011 учебного года, которые участвовали в технической подготовке и презентации второго издания (далее их фамилии приведены в алфавитном порядке): Алдриана Обая Муиса, Бах Нгок Тхань Конга, Чэнь Цзюньчэна, Девендру Гоял, Фикрила Бахри, Хасана Али Аскари, Харта Виджая, Хун Даи Тана, Ко Цзы Чуна, Ли Ин Конга, Питера Панди, Раймонда Хенди Сьюзанто, Сима Венлунга Рассела, Тан Хианг Тата, Тран Конг Хоанг, Юань Юаня и еще одного студента, который предпочел остаться анонимным;

- читателей корректуры: семерых студентов, обучавшихся на курсе CS3233 (выше их имена подчеркнуты), и Тай Вэньбинь;
- и последнее, но не менее важное: я хочу поблагодарить мою жену, Грейс Сурьяни, за то, что она позволила мне провести время за работой над очередным изданием книги, когда она была беременна нашим первым ребенком, Джейн Анджелиной Халим.

Благодарности к третьему изданию

От Стивена: еще раз я хочу поблагодарить:

- 2000 покупателей второго издания (по состоянию на 24 мая 2013 года). Спасибо :);
- помощников преподавателя курса CS3233 Национального университета Сингапура, участвовавших в проведении курса за последние два года: Харта Виджая, Тринь Туан Фуонг и Хуан Да;
- моих студентов, обучавшихся на курсе CS3233 во втором семестре 2011/2012 учебного года, которые участвовали в технической подготовке и презентации второго издания (далее их фамилии приведены в алфавитном порядке): Цао Шэна, Чуа Вэй Куан, Хань Юй, Хуан Да, Хуинь Нгок Тай, Ивана Рейнальдо, Джона Го Чу Эрна, Ле Вьет Тьена, Лим Чжи Цинь, Налина Иланго, Нгуен Хоанг Дуй, Нгуен Фи Лонга, Нгуен Куок Фонг, Паллава Шингаля, Пан Чжэнъяна, Пан Ян Хана, Сун Янью, Тан Ченг Йонг Десмонд, Тэй Вэньбинь, Ян Маньшена, Чжао Яна, Чжоу Имина и двух других студентов, которые предпочли остаться анонимными;



- читателей корректуры: шестерых студентов, обучавшихся на курсе CS3233 во втором семестре 2011/2012 учебного года (выше их имена подчеркнуты), и Хьюберта Тео Хуа Киана;
- моих студентов, обучавшихся на курсе CS3233 во втором семестре 2012/2013 учебного года, которые участвовали в технической подготовке и презентации второго издания (далее их фамилии приведены в алфа-

витном порядке): Арнольда Кристофера Короа, Цао Луу Куанг, Лим Пуай Линг Полин, Эрика Александра Квик Факсаа, Джонатана Дэррила Виджаи, Нгуен Тан Сы Нгуен, Нгуен Чыонг Дуй, Онг Мин Хуэй, Пан Юйсюань, Шубхам Гоял, Судханшу Хемку, Тан Бинбин, Трин Нгок Кхана, Яо Юцзянь, Чжао Юэ и Чжэн Найцзя;

- центр развития преподавания и обучения (CDTL) Национального университета Сингапура за предоставление начального финансирования для создания веб-сайта визуализации алгоритмов;
- мою жену Грейс Сурьяни и мою дочь Джейн Анджелину за их любовь.

Стивен и Феликс Халим
Сингапур, 24 мая 2013 г.

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии данной книги. Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Скачивание исходного кода

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com на странице с описанием соответствующей книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» и авторы книги очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Об авторах этой книги

Стивен Халим, доктор философии[1]
stevenhalim@gmail.com

Стивен Халим в настоящее время преподает в Школе программирования Национального университета Сингапура (SoC, NUS). Он ведет несколько курсов по программированию в университете Сингапура, начиная с основ методологии программирования, далее переходя к курсам среднего уровня, отноящимся к структурам данных и алгоритмам, а также курс «Олимпиадное программирование», в качестве учебного материала к которому используется эта книга. Стивен является наставником команд Национального университета Сингапура, участвующих в соревнованиях по программированию ACM ICPC, а также наставником сингапурской команды IOI. Будучи студентом, участвовал в нескольких региональных состязаниях ACM ICPC (Сингапур 2001, Айзу 2003, Шанхай 2004). На сегодняшний день он и другие наставники Национального университета Сингапура успешно подготовили две команды финалистов соревнований мирового уровня ACM ICPC World (2009–2010; 2012–2013). Также их студенты завоевали две золотые, шесть серебряных и семь бронзовых наград на олимпиаде IOI (2009–2012).

Стивен женат на Грейс Сурьяни Тиозо; супруги воспитывают дочь, Джейн Анджелину Халим.

Феликс Халим, доктор философии[2]
felix.halim@gmail.com

Феликс Халим получил степень доктора философии в Школе программирования Национального университета Сингапура (SoC, NUS). С точки зрения соревнований по программированию, у Феликса гораздо более яркая репутация, чем у его старшего брата. Он был участником IOI 2002 (представлял Индонезию). Команды ICPC, в которых он участвовал (в то

---

[1] Кандидатская диссертация на тему: «An Integrated White+Black Box Approach for Designing and Tuning Stochastic Local Search Algorithms» (Интегрированный подход «белого + черного ящика» для разработки и усовершенствования алгоритмов стохастического локального поиска), 2009.

[2] Кандидатская диссертация на тему: «Solving Big Data Problems: from Sequences to Tables and Graphs» (Решение задач, использующих большие данные: от последовательностей к таблицам и графам), 2012.

время он был участником соревнований от университета Бина Нусантара), заняли на региональных соревнованиях в Маниле (ACM ICPC Manila Regional 2003–2004–2005) 10-е, 6-е и 10-е места соответственно. Затем в последний год его команда, наконец, выиграла гаосюнские региональные соревнования (ACM ICPC Kaohsiung Regional 2006) и стала финалистом соревнований мирового уровня в Токио (ACM ICPC World Tokyo 2007), заняв 44-е место. Сегодня он активно участвует в матчах TopCoder Single Round Matches, и его самый высокий рейтинг – «желтый» пояс по программированию. В настоящий момент Феликс Халим работает в компании Google, г. Маунтин-Вью, США.

Список сокращений

A*: A со звездочкой
ACM: Assoc of Computing Machinery – Ассоциация вычислительной техники
AC: Accepted – принято
APSP: All-Pairs Shortest Paths – кратчайшие расстояния между всеми вершинами
AVL: Adelson-Velskii Landis (BST) – АВЛ-дерево (алгоритм Адельсон-Вельского и Ландиса)
BNF: Backus Naur Form – форма Бэкуса–Наура
BFS: Breadth First Search – поиск в ширину
BI: Big Integer – большое целое
BIT: Binary Indexed Tree – двоичное индексированное дерево
BST: Binary Search Tree – дерево двоичного поиска
CC: Coin Change – размен монет
CCW: Counter ClockWise – против часовой стрелки
CF: Cumulative Frequency – накопленная частота
CH: Convex Hull – выпуклая оболочка
CS: Computer Science – компьютерные науки
CW: ClockWise – по часовой стрелке
DAG: Directed Acyclic Graph – направленный ациклический граф
DAT: Direct Addressing Table – таблица с прямой адресацией
D&C: Divide and Conquer – «разделяй и властвуй»
DFS: Depth First Search – поиск в глубину
DLS: Depth Limited Search – поиск с ограничением глубины
DP: Dynamic Programming – динамическое программирование
DS: Data Structure – структура данных
ED: Edit Distance – расстояние редактирования
FIFO: First In First Out – принцип FIFO («первым пришел – первым ушел»)
FT: Fenwick Tree – дерево Фенвика
GCD: Greatest Common Divisor – НОД (наибольший общий делитель)
ICPC: Intl Collegiate Prog Contest – Международная студенческая олимпиада по программированию
IDS: Iterative Deepening Search – поиск с итеративным углублением
IDA*: Iterative Deepening A Star – итеративное углубление A*
IOI: Intl Olympiad in Informatics – международные олимпиады по информатике
IPSC: Internet Problem Solving Contest – интернет-конкурс по решению задач
LA : Live Archive [33]
LCA: Lowest Common Ancestor – самый низкий общий предок
LCM: Least Common Multiple – наименьшее общее кратное; НОК
LCP: Longest Common Prefix – наибольший общий префикс
LCS1: Longest Common Subsequence – наибольшая общая подпоследовательность

LCS2: Longest Common Substring – наибольшая общая подстрока
LIFO: Last In First Out – принцип LIFO («последним пришел – первым ушел»)
LIS: Longest Increasing Subsequence – наибольшая возрастающая подпоследовательность
LRS: Longest Repeated Substring – самая длинная повторяющаяся подстрока
LSB: Least Significant Bit – наименьший значащий бит
MCBM: Max Cardinality Bip Matching – максимальное по мощности паросочетание на двудольном графе
MCM: Matrix Chain Multiplication – умножение матричной цепи
MCMF: Min-Cost Max-Flow – максимальный поток минимальной стоимости
MIS: Maximum Independent Set – максимальное независимое множество
MLE: Memory Limit Exceeded – ошибка превышения лимита памяти
MPC: Minimum Path Cover – минимальное покрытие путями
MSB: Most Significant Bit – самый старший двоичный разряд
MSSP: Multi-Sources Shortest Paths – задача о кратчайших путях из заданных вершин во все
MST: Minimum Spanning Tree – минимальное остовное дерево
MWIS: Max Weighted Independent Set – независимое множество с максимальным весом
MVC: Minimum Vertex Cover – минимальное вершинное покрытие
OJ: Online Judge – «Онлайн-арбитр»
PE: Presentation Error – ошибка представления
RB: Red-Black (BST) – черно-красное (дерево двоичного поиска)
RMQ: Range Min (or Max) Query – запрос минимального (максимального) значения из диапазона
RSQ: Range Sum Query – запрос суммы диапазона
RTE: Run Time Error – ошибка времени выполнения
SSSP: Single-Source Shortest Paths – кратчайшие пути из заданной вершины во все остальные
SA: Suffix Array – массив суфиксов
SCC: Strongly Connected Component – компонент сильной связности
SPOJ: Sphere Online Judge – «Онлайн-арбитр» в Sphere
ST: Suffix Tree – дерево суффиксов
STL: Standard Template Library – стандартная библиотека шаблонов
TLE: Time Limit Exceeded – превышение лимита времени
USACO: USA Computing Olympiad – олимпиада по программированию в США
UVa: University of Valladolid [47] – университет Вальядолида
WA: Wrong Answer – неверный ответ
WF: World Finals – финальные соревнования на кубок мира

Глава 1

Введение

Я хочу участвовать в финальных соревнованиях на кубок мира ACM ICPC!

– Прилежный студент

1.1. Олимпиадное программирование

Основная идея в спортивном программировании такова: «Вам дают известные задачи в области компьютерных наук, решайте их как можно быстрее!»

Давайте прочитаем эту фразу более внимательно, обращая внимание на термины. Термин «известные задачи в области компьютерных наук» подразумевает, что в олимпиадном программировании мы имеем дело с *уже решенными* задачами, а не с задачами, требующими исследовательского подхода (где до сих пор нет решений). Эти задачи уже были решены ранее (по крайней мере, автором задач). «Решать их» подразумевает, что мы[1] должны углубить наши знания в области компьютерных наук. Мы должны достичь необходимого уровня, позволяющего создавать работающий код, который решает эти задачи, по крайней мере мы должны получить в установленные сроки тот же результат, что и автор задачи, на неизвестных нам тестовых данных, подготовленных разработчиком задачи[2]. Необходимость решить задачу «как можно быстрее» – это одна из целей подобных соревнований: ведь соревнования в скорости – в природе человека.

---

[1] Некоторые соревнования по программированию проводятся в командах, чтобы стимулировать командную работу, ведь инженеры-программисты обычно не работают в одиночку в реальной жизни.

[2] Публикуя постановку задачи, но не фактические тестовые данные для нее, олимпиадное программирование побуждает соревнующихся в решении задач проявлять креативность и аналитическое мышление, чтобы обдумать все возможные варианты решения задачи и протестировать свой код. В реальной жизни разработчикам программного обеспечения приходится изобретать тесты и много раз тестировать свое программное обеспечение, чтобы убедиться, что оно соответствует требованиям клиентов.

Пример. Архив задач университета Вальядолида [47], задача № 10 911 (Формирование команды для викторины).

Краткое описание задачи:
Пусть (x, y) – координаты дома, в котором живет студент, на двухмерной плоскости. Есть $2N$ студентов, которых мы хотим объединить в N групп. Пусть d_i – расстояние между домами двух студентов в группе i.
Формируем N групп таким образом, чтобы значение cost = $\sum_{i=1}^{N} d_i$ было минимальным.
Выведите минимальное значение cost. Ограничения: $1 \leqslant N \leqslant 8$ и $0 \leqslant x, y \leqslant 1000$.

Пример входных данных:
$N = 2$; Координаты $2N = 4$ дома: {1, 1}, {8, 6}, {6, 8} и {1, 3}.

Пример выходных данных:
cost = 4,83.

Сможете ли вы решить эту задачу?
Если да, сколько времени вам потребуется, чтобы написать работающий код?
Подумайте и постарайтесь не переворачивать эту страницу сию же секунду!

Рис. 1.1 ❖ Иллюстрация к задаче UVa 10911 – Forming Quiz Teams

Теперь спросите себя: к какому из описанных ниже программистов вы можете отнести себя? Обратите внимание, что если вам непонятен материал или терминология из этой главы, вы можете повторно вернуться к нему, прочитав эту книгу еще один раз.

- Неконкурентоспособный программист A (начинающий программист)
  Шаг 1: читает условия задачи и не понимает ее. (Эта задача для него новая.)
  Шаг 2: пытается написать какой-то код, в отчаянии смотрит на непонятные входные и выходные данные.
  Шаг 3: понимает, что все его попытки решения не зачтены (AC не получено), использует «жадный» алгоритм (см. раздел 3.4). Попарное

объединение студентов, чьи дома находятся на кратчайшем расстоянии между ними, дает неверный ответ (WA).
Предпринимает наивную попытку использовать полный перебор: использует возвратную рекурсию (см. раздел 3.2) и комбинации из всех возможных пар, что приводит его к превышению лимита времени (TLE).

- Неконкурентоспособный программист B (сдается)
  Шаг 1: читает условия задачи и вспоминает, что он где-то читал об этом раньше.
  Но также он помнит, что не научился решать такие задачи…
  Он не знает о решении, относящемся к области динамического программирования (DP) (см. раздел 3.5).
  Шаг 2: пропускает эту задачу и читает другую из списка задач, предложенных на соревнованиях.
- Все еще неконкурентоспособный программист C (решает задачу медленно)
  Шаг 1: читает условия задачи и понимает, что это сложная задача: поиск идеального соответствия с минимальным весом на небольшом произвольном взвешенном графе. Однако, поскольку объем файла входных данных невелик, эту задачу можно решить средствами динамического программирования (DP). Состояние в динамическом программировании – это **битовая маска**, описывающая состояние соответствия, и при сопоставлении не соответствующих условию проживания на минимальном расстоянии студентов *i* и *j* включаются два бита *i* и *j* битовой маски (раздел 8.3.1).
  Шаг 2: пишет код процедуры ввода-вывода, использует нисходящий рекурсивный метод динамического программирования, пишет тесты, отлаживает программу…
  Шаг 3: через 3 часа его решение получает оценку «зачтено» (AC) – оно проходит все тесты с использованием секретного набора тестовых данных.
- Конкурентоспособный программист D
  Завершает все шаги, предпринятые неконкурентоспособным программистом C, за время, не превышающее 30 минут.
- Высококонкурентоспособный программист E
  Высококонкурентоспособный конкурентоспособный программист (например, обладатель «красного рейтинга» на сайте TopCoder [32]) решит эту «хорошо известную» задачу за 15 минут.

Мы хотим обратить ваше внимание на то, что успехи в спортивном программировании – не конечная цель, а лишь средство достижения цели. Истинная конечная цель состоит в том, чтобы подготовить ученых в области информатики и программистов, которые создают более качественное программное обеспечение и ставят перед собой очень серьезные исследовательские задачи. Учредители студенческого командного чемпионата мира по программированию (ICPC) [66] придерживаются этой концепции, и мы, авторы, солидарны с ними.

Написав эту книгу, мы внесли свой вклад в подготовку нынешнего и будущих поколений к тому, чтобы стать более конкурентоспособными в решении хорошо известных задач из области программирования, часто предлагаемых в заданиях последних состязаний ICPC и на Международной олимпиаде по информатике (IOI).

Упражнение 1.1.1. Описанная выше стратегия неконкурентоспособного программиста A, использующего «жадные» алгоритмы, дает правильный результат для тестового примера, показанного на рис. 1.1. Пожалуйста, приведите *лучший* контрпример.

Упражнение 1.1.2. Проанализируйте временную сложность решения задачи методом полного перебора, представленным неконкурентоспособным программистом A и приведенным выше, чтобы понять, почему оно получает вердикт TLE.

Упражнение 1.1.3*. На самом деле продуманное решение, использующее возвратную рекурсию, *с ограничением*, все же поможет решить эту задачу. Решите эту задачу без использования таблицы динамического программирования.

1.2. Как стать конкурентоспособным

Если вы хотите стать похожими на конкурентоспособных программистов D или E из приведенного выше примера – то есть если вы хотите, чтобы вас выбрали (обычно отбор происходит на уровне стран, отобранные кандидаты становятся членами сборной команды) для участия в соревнованиях IOI [34], или же вы хотите стать одним из членов команды, представляющей ваш университет в ICPC [66] (соревнования первого и второго уровней проводятся внутри страны, далее идут региональные и мировые турниры), либо вы желаете отличиться в других соревнованиях по программированию – то эта книга определенно для вас!

В последующих главах вы узнаете все о задачах, предполагающих использование структур данных и алгоритмов – начиная с элементарных задач, переходя к задачам средней сложности и завершая сложными задачами[1], которые часто предлагались в недавно прошедших олимпиадах по программированию. Эти задачи были собраны из многих источников [50, 9, 56, 7, 40, 58, 42, 60, 1, 38, 8, 59, 41, 62, 46] (см. рис. 1.4). Вы не только получите представление о структурах данных и алгоритмах, но и узнаете, как их эффективно реализовать и применять в решении олимпиадных задач. Кроме того, книга дает вам много советов по программированию, основанных на нашем собственном опыте, который может быть полезен при участии в соревнованиях по программированию. Мы начнем эту книгу с нескольких общих советов.

---

[1] Восприятие материала, представленного в этой книге, зависит от вашей подготовки. Найдете ли вы представленный в ней материал сложным или среднего уровня сложности, зависит от ваших навыков программирования до прочтения этой книги.

1.2.1. Совет 1: печатайте быстрее!

Это не шутка! Хотя этому совету можно не придавать большого значения, поскольку соревнования ICPC и особенно IOI – это не соревнования в скорости печати, мы часто оказывались свидетелями ситуаций, когда команды ICPC, которые опускались на более низкие строчки в таблице рейтинга соревнований, представляли правильное решение всего лишь на несколько минут позже своих соперников; мы видели расстроенных участников соревнований IOI, которые упускали возможность получить баллы за правильно решенные задачи, просто не успев дописать код при решении задач методом перебора. Если вы и ваши соперники решаете одинаковое количество задач за отведенное время, это происходит благодаря вашим навыкам программирования (способности создавать лаконичный и надежный код) и… скорости, с которой вы набираете текст.

Попробуйте пройти тест на сайте http://www.typingtest.com – следуйте инструкциям, опубликованным на этом сайте. У Стивена скорость набора текста – около 85–95 знаков в минуту, а у Феликса – около 55–65 знаков в минуту. Если ваша скорость печати намного меньше этих цифр, отнеситесь к нашему первому совету серьезно!

Помимо возможности быстрого и правильного ввода букв и цифр, вам также необходимо ознакомиться с положением символов, часто используемых в языках программирования, таких как круглые скобки (), фигурные скобки {}, квадратные скобки [] и угловые скобки <>, точка с запятой ; и двоеточие :, одинарные кавычки ' ', используемые для обозначения символов, двойные кавычки " ", используемые для обозначения строк, амперсанд &, вертикальная черта (или «пайп») |, восклицательный знак ! и т. д.

В качестве небольшого упражнения попробуйте набрать код на C++, приведенный ниже, как можно быстрее.

```
#include <algorithm>                  // если вы не понимаете, как работает этот код на C++,
#include <cmath>                       // сначала прочитайте учебники по программированию...
#include <cstdio>
#include <cstring>
using namespace std;
           /* Формирование команд; ниже представлено решение задачи UVa 10911 */
                    // использование глобальных переменных – плохая практика при разработке
                                                                    // программного обеспечения,
int N, target;                       // но вполне подходит для олимпиадного программирования
double dist[20][20], memo[1 << 16];               // 1 << 16 = 2^16, заметим, что max N = 8

double matching(int bitmask) {                                      // состояние DP = bitmask
                                    // инициализируем 'memo' значением -1 в функции main
  if (memo[bitmask] > -0.5)                          // это состояние было вычислено ранее
    return memo[bitmask];                                          // просмотр таблицы memo
  if (bitmask == target)                             // все студенты уже разбиты на группы
    return memo[bitmask] = 0;                                      // значение cost равно 0

  double ans = 2000000000.0;                 // инициализируем переменную большим значением
```

```
  int p1, p2;
  for (p1 = 0; p1 < 2 * N; p1++)
    if (!(bitmask & (1 << p1)))
      break;                                          // найдем первый выключенный бит
  for (p2 = p1 + 1; p2 < 2 * N; p2++)                 // затем пытаемся сопоставить p1
    if (!(bitmask & (1 << p2)))                     // с другим битом p2, также выключенным
      ans = min(ans,                                  // вычисляем минимальное значение
                dist[p1][p2] + matching(bitmask | (1 << p1) | (1 << p2)));

  return memo[bitmask] = ans;                       // сохраняем результат в таблице memo
}

int main() {
  int i, j, caseNo = 1, x[20], y[20];
  // freopen("10911.txt", "r", stdin);                // подаем файл со входными данными,
                                                     // на стандартный ввод
  while (scanf("%d", &N), N) {                       // да, мы так можем :)
    for (i = 0; i < 2 * N; i++)
      scanf("%*s %d %d", &x[i], &y[i]);                          // '%*s' пропускает имена
    for (i = 0; i < 2 * N - 1; i++)                 // строим таблицу попарных расстояний
      for (j = i + 1; j < 2 * N; j++)               // вы когда-нибудь использовали 'hypot'?
        dist[i][j] = dist[j][i] = hypot(x[i] - x[j], y[i] - y[j]);

    // использование динамического программирования для поиска
    // идеального соответствия с минимальным весом
    // на небольшом произвольном взвешенном графе
    for (i = 0; i < (1 << 16); i++) memo[i] = -1.0;    // задаем значение -1 для всех ячеек
    target = (1 << (2 * N)) - 1;
    printf("Случай %d: %.2lf\n", caseNo++, matching(0));
} } // возвращаем 0;
```

Объяснение этого решения «динамическое программирование на битовых масках» приведено в разделах 2.2, 3.5 и 8.3.1. Не пугайтесь, если вы еще не до конца поняли, как решается данная задача.

1.2.2. Совет 2: быстро классифицируйте задачи

В ICPC участникам (командам) предоставляется несколько задач (обычно 7–14 задач) разных типов. По нашим наблюдениям за тем, как проходили соревнования ICPC в странах Азиатского региона, мы можем разбить задачи на категории и примерно определить, какой процент от общего числа задач составляют задачи каждой категории (см. табл. 1.1).

В IOI участникам дается 6 заданий, которые они должны решить в течение двух туров, по три задачи на каждый тур (на соревнованиях 2009–2010 гг. давалось 8 заданий, которые нужно было решить в течение двух дней). Эти задачи в основном относятся к строкам 1–5 и 10 в табл. 1.1; число задач, относящихся к строкам 6–10 в табл. 1.1, в олимпиаде IOI значительно меньше. Более подробно содержание задач IOI раскрыто в опубликованной программе IOI 2009 года [20] и классификации задач IOI за 1989–2008 гг. [67].

Таблица 1.1. Классификация задач, предложенных на последних олимпиадах по программированию ACM ICPC (Азия)

№	Класс задач	В этой книге	Число предложенных задач
1	AdHoc	Раздел 1.4	1–2
2	Полный перебор (итеративный/рекурсивный)	Раздел 3.2	1–2
3	«Разделяй и властвуй»	Раздел 3.3	0–1
4	«Жадные» алгоритмы (как правило, оригинальные)	Раздел 3.4	0–1
5	Динамическое программирование (как правило, оригинальные)	Раздел 3.5	1–3
6	Графы	Глава 4	1–2
7	Математические задачи	Глава 5	1–2
8	Обработка строк	Глава 6	1
9	Вычислительная геометрия	Глава 7	1
10	Сложные/редкие темы	Главы 8–9	1–2
Всего			8–17 (≈⩽ 14)

Классификация, приведенная в табл. 1.1, взята из [48] и никак не претендует на полноту и завершенность. Некоторые методы (например, сортировка) исключены из классификации как «тривиальные»; обычно они используются только в качестве «подпрограмм» в более серьезных задачах. Мы также не включаем в классификацию раздел «рекурсия», так как она входит в такие категории задач, как возвратная рекурсия или динамическое программирование. Мы также опускаем раздел «структуры данных», поскольку использование эффективной структуры данных можно считать неотъемлемой частью решения более сложных задач. Конечно, решение задач не всегда может быть сведено к единственному методу: задача может быть разделена на несколько подзадач, относящихся к разным классам. Например, алгоритм Флойда–Уоршелла можно классифицировать и как решение задачи нахождения кратчайших расстояний между всеми вершинами графа (см. раздел 4.5), и как решение задачи с использованием алгоритмов динамического программирования (см. раздел 3.5). Алгоритмы Прима и Краскала являются решением задачи построения минимального остовного дерева (см. раздел 4.3) и в то же время могут быть отнесены к «жадным» алгоритмам (см. раздел 3.4). В разделе 8.4 мы рассмотрим более сложные задачи, для решения которых требуется использовать несколько алгоритмов и/или структур данных.

В будущем эта классификация может измениться. Возьмем, к примеру, динамическое программирование. Этот метод не был известен до 1940-х годов и не входил в программы ICPC и IOI до середины 1990-х годов, но в настоящее время это один из обязательных видов задач на олимпиадах по программированию. Так, например, в финале чемпионата мира ICPC 2010 было предложено более трех задач, относящихся к этому классу (из 11 предлагавшихся для решения).

Однако наша главная цель – не просто сопоставить задачи и методы, необходимые для их решения, как это сделано в табл. 1.1. Ознакомившись с большин-

ством тем, изложенных в этой книге, вы сможете распределять задачи, относя их к одному из трех типов, как показано в табл. 1.2.

Таблица 1.2. Категории задач (в более компактной форме)

№	Категория задач	Уверенность и ожидаемая скорость решения
A	Я решал задачи такого типа	Я уверен, что смогу решить такую задачу (быстро)
B	Я где-то встречал такую задачу	Но я знаю, что пока не могу решить ее
C	Я никогда не встречал задач такого типа	См. обсуждение ниже

Чтобы быть *конкурентоспособным*, то есть занимать высокие места на олимпиадах по программированию, вы должны отнести абсолютное большинство предложенных задач к типу A и минимизировать количество задач, которые вы относите к типу B. Вам необходимо приобрести прочные знания в области алгоритмов и совершенствовать свои навыки программирования, чтобы легко решать многие классические задачи. Однако, чтобы выиграть олимпиаду по программированию, вам также необходимо совершенствовать и оттачивать навыки решения задач (например, умение сводить решение предложенной задачи к решению известных задач, видеть тонкости и «хитрости» в задаче, использовать нестандартные подходы к решению задач и т. д.) – так, чтобы вы (или ваша команда) смогли найти решение для сложных, нетривиальных задач типа C, участвуя в региональных и всемирных олимпиадах IOI или ICPC.

Таблица 1.3. Упражнение: классифицируйте эти задачи с сайта университета Вальядолида (UVa)

№ UVa	Название	Категория задачи	Подсказка
10360	Rat Attack	Полный перебор или динамическое программирование	Раздел 3.2
10341	Solve It		Раздел 3.3
11292	Dragon of Loowater		Раздел 3.4
11450	Wedding Shopping		Раздел 3.5
10911	Forming Quiz Teams	Динамическое программирование с использованием битовой маски	Раздел 8.3.1
11635	Hotel Booking		Раздел 8.4
11506	Angry Programmer		Раздел 4.6
10243	Fire! Fire!! Fire!!!		Раздел 4.7.1
10717	Mint		Раздел 8.4
11512	GATTACA		Раздел 6.6
10065	Useless Tile Packers		Раздел 7.3.7

Упражнение 1.2.1. Прочитайте условия задач с сайта университета Вальядолида [47], перечисленных в табл. 1.3, и определите, к каким категориям они относятся (для двух задач в данной таблице мы уже проделали это упражнение). После прочтения нашей книги вам будет легко выполнить такое упражнение – в книге обсуждаются все методы, необходимые для решения этих задач.

1.2.3. Совет 3: проводите анализ алгоритмов

После того как вы разработали алгоритм для решения конкретной задачи на олимпиаде по программированию, вы должны задать себе вопрос: учитывая ограничения по объему входных данных (обычно указанные в содержательной постановке задачи), может ли разработанный алгоритм с его временной сложностью и пространственной сложностью (сложностью по памяти) уложиться в отведенный лимит времени и памяти, указанный для этой задачи?

Иногда существует несколько способов решения задачи. Некоторые подходы могут оказаться неверными, другие – недостаточно быстрыми, а третьи могут выйти за пределы разумной оптимизации. Хорошая стратегия – провести мозговой штурм, перебирая множество подходящих алгоритмов, а затем выбрать простейшее решение, которое работает (то есть работает достаточно быстро, чтобы преодолеть ограничение по времени и памяти и при этом все же дать правильный ответ)[1].

Современные компьютеры достаточно мощны и могут выполнять[2] до 100 млн (или 10^8; 1 млн = 1 000 000) операций за несколько секунд. Вы можете использовать эту информацию, чтобы определить, уложится ли ваш алгоритм в отведенное время. Например, если максимальный размер входных данных n равен 100 КБ (или 10^5; 1 КБ = 1000), а ваш текущий алгоритм имеет временную сложность $O(n^2)$, то простейшие вычисления покажут, что $(100\text{ КБ})^2$ или 10^{10} – это очень большое число, и ваш алгоритм будет отрабатывать за время порядка сотни секунд. Таким образом, вам нужно будет разработать более быстрый (и при этом правильный) алгоритм для решения задачи. Предположим, вы нашли тот, который работает с временной сложностью $O(n \log_2 n)$. Теперь расчеты показывают, что $10^5 \log_2 10^5$ – это всего лишь $1{,}7 \times 10^6$, и здравый смысл подсказывает, что алгоритм (который теперь должен работать менее чем за секунду), скорее всего, работает достаточно быстро, чтобы уложиться во временные ограничения.

При определении того, подходит ли ваше решение, ограничения задачи играют не меньшую роль, чем временная сложность вашего алгоритма. Предположим, что вы можете разработать только относительно простой алгоритм, для которого легко написать код, но который работает с кошмарной временной сложностью $O(n^4)$. Это может показаться неподходящим решением, но если $n \leqslant 50$, то вы действительно решили задачу. Вы можете реализовать свой алгоритм $O(n^4)$, поскольку 50^4 составляет всего 6,25 млн, и ваш алгоритм должен отрабатывать примерно за секунду.

Обратите внимание, однако, что порядок сложности не обязательно указывает на фактическое количество операций, которые будет выполнять ваш ал-

---

[1] **Обсуждение:** это действительно так – на олимпиадах по программированию выбор простейшего алгоритма, который работает, крайне важен для успеха в соревнованиях. Тем не менее на занятиях в аудитории, где нет жестких временных ограничений, полезно потратить больше времени на решение определенной задачи с использованием наилучшего алгоритма. Поступая таким образом, мы повышаем уровень своей подготовки. Если в будущем мы столкнемся с более сложной версией задачи, у нас будет больше шансов получить и реализовать правильное решение!

[2] Используйте приведенную здесь цифру для приблизительных расчетов. Значение производительности, разумеется, будет различаться для разных компьютеров.

горитм. Если каждая итерация включает в себя большое количество операций (много вычислений с плавающей точкой или значительное число итераций циклов), или же если ваша реализация имеет высокие «накладные расходы» при выполнении (множество повторяющихся циклов, несколько проходов или даже высокие затраты на операции ввода-вывода либо исполнение программы), ваш код может работать медленнее, чем ожидалось. Однако обычно это не представляет серьезной проблемы, поскольку авторы задачи устанавливают ограничения по времени таким образом, чтобы хорошо написанный код, реализующий алгоритм с подходящей сложностью по времени, получил оценку «зачтено» (AC).

Анализируя сложность вашего алгоритма с учетом определенного размера входных данных и указанных ограничений по времени и памяти, вы можете выбрать правильную стратегию: решить, следует ли вам пытаться реализовать свой алгоритм (что отнимет драгоценное время в соревнованиях ICPC и IOI), попытаться улучшить свой алгоритм или же переключиться на другие задачи, предложенные на соревнованиях.

Как уже упоминалось в предисловии к этой книге, мы не будем подробно обсуждать концепцию алгоритмического анализа. Мы предполагаем, что у вас уже есть этот элементарный навык. Существует множество справочников и книг (например, «Введение в алгоритмы» («Introduction to Algorithms») [7], «Разработка алгоритмов» («Algorithm Design») [38], «Алгоритмы» («Algorithms») [8], и т. д.), которые помогут вам понять следующие обязательные понятия/методы в алгоритмическом анализе:

- базовый анализ сложности по времени и по памяти для итерационных и рекурсивных алгоритмов:
  - алгоритм с k вложенными циклами, состоящими примерно из n итераций, имеет сложность по времени $O(n^k)$;
  - если у вас имеется рекурсивный алгоритм с b рекурсивными вызовами на уровень и глубина рекурсии составляет L уровней, то сложность такого алгоритма будет приблизительно $O(b^L)$, однако подобная оценка – лишь грубая верхняя граница. Фактическая сложность алгоритма будет зависеть от того, какие действия выполняются на каждом уровне и возможно ли упрощение;
  - алгоритм динамического программирования или другая итерационная процедура, которая обрабатывает двумерную матрицу $n \times n$, затрачивая $O(k)$ на ячейку, отрабатывает за время $O(k \times n^2)$. Это более подробно объясняется в разделе 3.5;
- более продвинутые методы анализа:
  - докажите правильность алгоритма (это особенно важно для «жадных» алгоритмов, о которых идет речь в разделе 3.4), чтобы свести к минимуму вероятность получения оценки тестирующей системы «Неправильный ответ» (WA) для вашей задачи;
  - выполните амортизационный анализ (в качестве примера см. главу 17 в [7]). Амортизационный анализ, хотя и редко применяется на олимпиадах по программированию, позволяет свести к минимуму вероятность получения оценки тестирующей системы «Превышение лимита времени» (TLE) или, что еще хуже, случаи, когда вы отвергаете

свой алгоритм из-за того, что он слишком медленный, и бросаете решение этой задачи, переключаясь на другую, хотя на самом деле ваш алгоритм работает достаточно быстро;
- выполните анализ алгоритма на основе выходных данных, поскольку время работы алгоритма также зависит от размера выходных данных, – это снизит вероятность того, что тестирующая система выдаст вердикт «Превышение лимита времени» (TLE) для вашего решения. Например, алгоритм поиска строки длиной m в длинной строке с помощью дерева суффиксов (которое уже построено) будет работать за время $O(m+occ)$. Время выполнения этого алгоритма зависит не только от размера входных данных m, но и от размера выходых данных – количества вхождений occ (более подробно об этом рассказывается в разделе 6.6);

❍ знание следующих ограничений:
- $2^{10} = 1024 \approx 10^3$, $2^{20} = 1\,048\,576 \approx 10^6$;
- 32-разрядные целые числа со знаком (`int`) и 64-разрядные целые числа со знаком (`long long`) имеют верхние пределы $2^{31} - 1 \approx 2\times10^9$ (что позволяет работать с числами, ограничивающимися приблизительно 9 десятичными разрядами) и $2^{63} - 1 \approx 9\times10^{18}$ (что позволяет работать с числами, ограничивающимися приблизительно 18 десятичными разрядами) соответственно;
- беззнаковые целые можно использовать, если требуются только неотрицательные числа. 32-разрядные целые числа без знака (`unsigned int`) и 64-разрядные целые числа без знака (`unsigned long long`) имеют верхние пределы $2^{32} - 1 \approx 4\times10^9$ и $2^{64} - 1 \approx 1{,}8\times10^{19}$ соответственно;
- если вам нужно хранить целые числа, значения которых превосходят 2^{64}, используйте методы работы с большими числами (см. раздел 5.3);
- число перестановок для множества из n элементов равно $n!$, число комбинаций для такого множества равно 2^n;
- наилучшая сложность по времени алгоритма сортировки на основе сравнения составляет $\Omega(n\log_2 n)$;
- обычно сложность по времени алгоритмов $O(n\log_2 n)$ вполне приемлема для решения большинства олимпиадных задач;
- наибольший размер входных данных для типичных задач олимпиады по программированию не должен превышать 1 млн. Кроме того, нужно принимать во внимание, что «узким местом» будет время чтения входных данных (процедура ввода-вывода);
- среднестатистический процессор, выпущенный в 2013 году, выполняет около 100 млн = 10^8 операций за несколько секунд.

Многие начинающие программисты пропускают этот этап и сразу же начинают реализовывать первый (наивный) алгоритм, пришедший им в голову, впоследствии убеждаясь, что выбранная структура данных или алгоритм недостаточно эффективны (или неверны). Наш совет всем участникам ICPC[1]: воз-

[1] В отличие от задач, предлагаемых на олимпиадах ICPC, задачи, которые решаются на IOI, обычно имеют нескольких возможных решений (частичных или полных), каждое из которых имеет различную временную сложность и оценивается таким

держитесь от написания кода, пока не убедитесь, что ваш алгоритм работает правильно и достаточно быстро.

Чтобы дать вам некоторые общие ориентиры временной сложности различных алгоритмов и помочь определить, что означает «достаточно быстро» в вашем случае, мы привели некоторые примеры в табл. 1.4. Подобные цифры также можно найти во многих других книгах по структурам данных и алгоритмам. Эта таблица составлена *участником олимпиады по программированию* с учетом контекста и специфики решения задач. Обычно ограничения размера входных данных приводятся в содержательной постановке задачи. Предполагая, что типичный процессор может выполнить 100 млн операций примерно за 3 секунды (что является стандартным ограничением по времени в большинстве задач, опубликованных на сайте университета Вальядолида (UVa) [47]), мы можем определить «худший» алгоритм, который все еще может уложиться в эти пределы времени. Обычно самый простой алгоритм имеет наихудшую временную сложность, но если он работает достаточно быстро, чтобы уложиться в отведенное время, просто используйте его!

Таблица 1.4. Эмпирическое правило определения наихудшей сложости по времени для алгоритма, получающего оценку жюри «зачтено» (AC), при различных объемах входных данных n при прохождении одного и того же теста (при условии что ваш процессор позволяет выполнять 100 млн операций за 3 с)

n	**Наихудшая сложость алгоритма по времени**	**Комментарий**
⩽ [10–11]	$O(n!)$, $O(n^6)$	Например: подсчет перестановок (см. раздел 3.2)
⩽ [15–18]	$O(2^n \times n^2)$	Например: решение задачи коммивояжера методами динамического программирования (ДП) (см. раздел 3.5.2)
⩽ [18–22]	$O(2^n \times n)$	Например: задачи динамического программирования с использованием операций с битовой маской (см. раздел 8.3.1)
⩽ 100	$O(n^4)$	Например: трехмерная задача динамического программирования (DP) + $O(n)$loop, ${}_nC_{k=4}$,
⩽ 400	$O(n^3)$	Например: алгоритм Флойда–Уоршелла (см. раздел 4.5)
⩽ 2K	$O(n^2 \log_2 n)$	Например: вложенные циклы с глубиной вложения = 2 + структуры данных на деревьях (см. раздел 2.3)
⩽ 10K	$O(n^2)$	Например: сортировка пузырьком/выбором/вставкой (см. раздел 2.2)
⩽ 1M	$O(n \log_2 n)$	Например: сортировка слиянием, построение дерева отрезков (см. раздел 2.3)
⩽ 100M	$O(n)$, $O(\log_2 n)$, $O(1)$	Большинство сложностей на олимпиадах по программированию возникают в случае $n \leqslant 1M$ («узкое место» – операции ввода-вывода)

образом, что за решение каждой из частей задачи начисляются баллы. Чтобы получить драгоценные баллы, можно попробовать решить задачу «в лоб», набрав таким образом несколько баллов и получив возможность лучше понять задачу. При этом за представленное «медленное» решение жюри не снимет баллы, поскольку в соревнованиях IOI скорость не является одним из главнейших факторов при оценке решения задач. Решив задачу перебором, постепенно улучшайте ваше решение, чтобы получить больше очков.

Упражнение 1.2.2. Ответьте на следующие вопросы, используя имеющиеся у вас знания о классических алгоритмах и их временной сложности. После того как вы прочитаете эту книгу до конца, попробуйте выполнить данное упражнение снова.

1. Существует n веб-страниц ($1 \leqslant n \leqslant 10M$). Каждая i-я веб-страница имеет рейтинг страницы r_i. Вы хотите выбрать 10 страниц с наивысшим рейтингом. Какой из описанных методов будет работать лучше:
   a) вы загрузите рейтинги всех n веб-страниц в память, отсортируете (см. раздел 2.2) их в порядке убывания рейтинга, взяв первые 10;
   b) вы используете очередь с приоритетом («кучу») (см. раздел 2.3).
2. Для заданной целочисленной матрицы Q, имеющей размерность $M \times N$ ($1 \leqslant M, N \leqslant 30$), определите, существует ли ее подматрица размера $A \times B$ ($1 \leqslant A \leqslant M$, $1 \leqslant B \leqslant N$), для которой среднее значение элементов матрицы mean(Q) = 7. Вы:
   a) построите все возможные подматрицы и проверите выполнение условия mean(Q) = 7 для каждой из подматриц. Временная сложность этого алгоритма составит $O(M^3 \times N^3)$;
   b) построите все возможные подматрицы, но реализуете алгоритм, имеющий временную сложность $O(M^2 \times N^2)$, используя следующий метод: _____.
3. Пусть имеется список L, состоящий из $10K$ целых чисел, и вам нужно часто вычислять значение суммы элементов списка, начиная с i-го и заканчивая j-м, т. е. `sum(i, j)`, где `sum(i, j)` = `L[i] + L[i + 1] + ... + L[j]`. Какую структуру данных нужно при этом использовать:
   a) простой массив (см. раздел 2.2);
   b) простой массив, предварительно обработанный с использованием метода динамического программирования (см. разделы 2.2 и 3.5);
   c) сбалансированное двоичное дерево поиска (см. раздел 2.3);
   d) двоичную кучу (см. раздел 2.3);
   e) дерево отрезков (см. раздел 2.4.3);
   f) двоичное индексированное дерево (дерево Фенвика) (см. раздел 2.4.4);
   g) суффиксный массив (см. раздел 6.6.2) или альтернативный вариант, суффиксный массив (см. раздел 6.6.4).
4. Для заданного множества S из N точек, случайно разбросанных по 2D-плоскости ($2 \leqslant N \leqslant 1000$), найдите две точки, принадлежащие множеству S, которые имеют наибольшее евклидово расстояние между ними. Подходит ли для решения данной задачи алгоритм полного перебора с временной сложностью $O(N^2)$, который перебирает все возможные пары точек:
   a) да, полный перебор подходит;
   b) нет, мы должны найти другой вариант решения. Мы должны использовать следующий метод: _____ .

5. Вы должны вычислить кратчайший путь между двумя вершинами на взвешенном ориентированном ациклическом графе (Directed Acyclic Graph, DAG), для которого выполняются условия $|V|, |E| \leqslant 100K$. Алгоритм(ы) какого типа можно использовать на олимпиаде по программированию (то есть с ограничением по времени приблизительно 3 секунды):
   a) динамическое программирование (см. разделы 3.5, 4.2.5 и 4.7.1);
   b) поиск в ширину (см. разделы 4.2.2 и 4.4.2);
   c) алгоритм Дейкстры (см. раздел 4.4.3);
   d) алгоритм Форда–Беллмана (см. раздел 4.4.4);
   e) алгоритм Флойда–Уоршелла (см. раздел 4.5).
6. Какой алгоритм создает список первых $10K$ простых чисел, обладая при этом лучшей временной сложностью (см. раздел 5.5.1):
   a) решето Эратосфена (см. раздел 5.5.1);
   b) проверка истинности выражения `isPrime(i)` для каждого числа $i \in [1..10K]$ (см. раздел 5.5.1).
7. Вы хотите проверить, является ли факториал числа n, то есть $n!$, числом, которое делится без остатка на целое число m. $1 \leqslant n \leqslant 10\,000$. Как вы выполните эту проверку:
   a) напишете `n! % m == 0`;
   b) наивный подход, приведенный выше, не будет работать, вы используете следующий метод: _____ (см. раздел 5.5.1).
8. Задание аналогично вопросу **4**, но с большим набором точек: $N \leqslant 1M$ – и одним дополнительным ограничением: точки случайным образом разбросаны по 2D-плоскости. Тогда:
   a) все еще можно использовать полный перебор, упомянутый в вопросе 3;
   b) наивный подход, приведенный выше, не будет работать, вы используете следующий метод: _____ (см. раздел 7.3.7).
9. Вы хотите найти и перечислить все вхождения подстроки P (длиной m) в (длинную) строку T (длины n). Ограничения длины строк: n и m имеют максимум $1M$ символов. Тогда:
   a) вы используете следующий фрагмент кода на C++:

```
for (int i = 0; i < n; i++) {
  bool found = true;
  for (int j = 0; j < m && found; j++)
    if (i + j >= n || P[j] != T[i + j]) found = false;
  if (found) printf("P is found at index %d in T\n", i);
}
```

   b) наивный подход, приведенный выше, не будет работать, и вы используете следующий метод: _____ (см. раздел 6.4 или 6.6).