АЛГОРИТМЫ НА ПРАКТИКЕ

РЕШЕНИЕ РЕАЛЬНЫХ ЗАДАЧ

ДАНИЭЛЬ ЗИНГАРО

no starch press

ББК 32.972.2-018
УДК 004.021
З-63

Зингаро Даниэль

З-63 Алгоритмы на практике. — СПб.: Питер, 2023. — 432 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-1853-3

«Алгоритмы на практике» научат решать самые трудные и интересные программистские задачи, а также разрабатывать собственные алгоритмы. В качестве примеров для обучения взяты реальные задания с международных соревнований по программированию. Вы узнаете, как классифицировать задачи, правильно подбирать структуру данных и выбирать алгоритм для решения. Поймете, что выбор структуры данных — будь то хеш-таблица, куча или дерево — влияет на скорость выполнения программы и на эффективность алгоритма. Разберетесь, как применять рекурсию, динамическое программирование, двоичный поиск.

Никакого условного псевдокода, все примеры сопровождаются исходным кодом на языке Си с подробными объяснениями.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.972.2-018
УДК 004.021

ISBN 978-1718500808 англ.

ISBN 978-5-4461-1853-3

Краткое содержание

Оглавление

ОБ АВТОРЕ

Даниэль Зингаро — доцент кафедры информатики в Университете Торонто, он неоднократно награждался за выдающиеся успехи в преподавании. Основная область его научных интересов — особенности обучения компьютерным наукам: он исследует, как студенты усваивают (а иногда не усваивают) материал в сфере computer science.

О НАУЧНОМ РЕДАКТОРЕ

Ларри Юли Чжан — доцент кафедры информатики в Университете Торонто. Область его преподавательских и исследовательских интересов включает алгоритмы, структуры данных, операционные системы, компьютерные сети, социальные сети и компьютерное образование. Он состоял в программных комитетах конференций ACM SIGCSE, ACM ITiCSE и WCCCE.

Предисловие

Начинающему теннисисту сложно послать мяч в нужное место корта (особенно бэкхендом). Только спустя месяцы практики начинает раскрываться притягательность этого вида спорта. Технический арсенал теннисиста постепенно расширяется — осваивается резаный бэкхенд, удар над головой, укороченный удар. Стратегия игрока переходит на более высокий уровень абстракции, где возможны выход к сетке с подачи, выход к сетке после резаного удара, игра на задней линии. Постепенно вырабатывается интуитивное понимание того, какие приемы и стратегии будут более эффективными против разных игроков, поскольку не существует универсального рецепта победы над любым соперником.

Программирование можно сравнить с теннисом. Начинающему программисту сложно доходчиво «объяснить» компьютеру, как нужно реализовать то или иное решение задачи. Но стоит превзойти уровень белого пояса, и работа с задачами начинает приносить удовольствие. Прежде всего, какой подход к решению выбрать? Хотя универсального средства эффективного решения всех задач не существует, есть надежные продвинутые инструменты и стратегии: хеш-таблицы, деревья поиска, рекурсия, мемоизация, динамическое программирование, поиск по графу и т. д. А опытному глазу многие задачи и алгоритмы сами указывают, какие инструменты подходят для них лучше всего. Ваш алгоритм выполняет повторяющийся поиск или минимальные вычисления? Ускорьте его с помощью хеш-таблицы или min-кучи соответственно. Общее решение основной задачи можно выстроить из вторичных решений ее подзадач? Используйте рекурсию! Эти подзадачи пересекаются? Ускорьте алгоритм с помощью мемоизации!

Будь то теннис или программирование, не получится перейти на более высокий уровень без двух вещей: практики и хорошего наставника. В случае программирования оба условия выполнят книга *«Алгоритмы на практике»* и ее автор Даниэль Зингаро. Он познакомит вас со всеми упомянутыми концепциями, но при этом не ограничится простым их перечнем. Под руководством Зингаро вы на примерах практических задач научитесь грамотному применению в работе правильных алгоритмических инструментов. Все это вы освоите с помощью книги, написанной понятным языком и с юмором. Удачи вам в решении задач!

Тим Рафгарден
Нью-Йорк, NY
Май 2020

Благодарности

Работа с ребятами из No Stratch Press была абсолютной идиллией. Они чрезвычайно сосредоточены на издании книг, помогающих читателям учиться. Здесь я нашел единомышленников! Лиз Чедвик (Liz Chadwick) поддержала мою книгу с самого начала (и не поддержала другую, за что я ей признателен). Было удовольствием работать с Алексой Фрид (Alex Freed), которая редактировала рукопись. Она терпелива, добра и не просто исправляла ошибки, а стремилась помочь мне улучшить текст. Я благодарю всех, кто участвовал в процессе создания книги, включая литературного редактора Дэвида Кузенса (David Couzens), выпускающего редактора Кэсси Андрэдис (Kassie Andreadis), креативного директора Дерека Йи (Derek Yee) и дизайнера обложки Роба Гейла (Rob Gale).

Я выражаю признательность Университету Торонто за поддержку моего труда. Спасибо Ларри Чжану (Larry Zhang), научному редактору, за внимательную вычитку рукописи. В течение нескольких лет я преподавал совместно с ним, и именно наше сотрудничество помогло мне выработать правильный образ мышления касательно алгоритмов и того, как обучать их использованию.

Благодарю Тима Рафгардена (Tim Roughgarden) за предисловие к моей книге. Книги и видео Тима представляют собой образцы ясности и наглядности, к которым нам нужно стремиться, рассказывая людям об алгоритмах.

Благодарю моих коллег Яна Варенхольда (Jan Vahrenhold), Махику Путане (Mahika Phutane) и Нааз Сибия (Naaz Sibia) за их рецензии на черновой вариант книги.

Спасибо всем авторам использованных в книге задач, а также участникам соревнований по программированию. Выражаю признательность администраторам

ресурса DMOJ за их поддержку моей работы. Отдельная благодарность — Тюдору Бриндусу (Tudor Brindus) и Раду Погонариу (Radu Pogonariu) за их помощь в выборе и доработке задач.

Спасибо моим родителям за то, что взяли на себя все, буквально все, и просили меня об одном — учиться.

Я благодарю Дояли, мою спутницу, за заботу и за то, что часть времени, которое мы могли бы провести вместе, она пожертвовала на написание моей книги.

В завершение я хочу выразить признательность всем вам за то, что читаете эту книгу и стремитесь к знаниям.

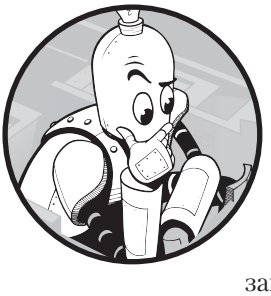
От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

Введение



Я предполагаю, что вы знакомы с такими языками, как Си, C++, Java или Python... и разбираетесь в теме. Тяжело объяснять людям, незнакомым с программированием, почему написание кода для решения задач является настолько захватывающим.

Я также надеюсь, что вы готовы поднять свои навыки программирования на новый уровень. Для меня будет честью помочь вам в этом.

Я мог бы начать с разъяснения всяких изощренных техник, рассказывая о том, какие они полезные, и сравнивая их с другими изощренными техниками. Но я не буду этого делать. Такие знания очень недолго хранятся в памяти, ожидая возможности применения на практике (которая порой так и не появляется).

Вместо этого я на протяжении всей книги буду ставить конкретные трудные задачи. Надеюсь, что вы не сможете справиться с ними сразу, используя знакомые вам подходы. Вы — программист. Вам нужно решать задачи. Пришло время освоить хитрые техники. Книга построена на постановке сложных задач с их последующим решением, при этом вы будете пополнять имеющиеся у вас знания.

Здесь вы не увидите типичных задачек из учебников, не будете искать оптимальный путь умножения последовательности матриц или вычисления чисел Фибоначчи. Я обещаю, что вы не будете решать и головоломку ханойской башни. Существует множество прекрасных учебников, где все это прописано, но я подозреваю, что немногих из вас такие головоломки действительно мотивируют.

Мой подход предполагает использование задач, которые вы ранее вряд ли встречали. Каждый год тысячи людей участвуют в соревнованиях по программированию, и для этих соревнований постоянно требуются новые задачи, чтобы оценивать реальные навыки участников, а не способность быстрее всех переделывать старое на новый лад или гуглить решение. Эти задачи удивительны, они базируются на классических подходах, но вносят в них изменения и интересный контекст, подталкивая людей к поиску новых решений. В таких задачах заключается практически безграничный объем знаний по программированию и вычислениям.

Начнем с основ. *Структура данных* — это способ организации данных для быстрого выполнения нужных операций. *Алгоритм* — это последовательность шагов при решении задачи. Иногда можно создать быстрый алгоритм без применения сложных структур данных. В других же случаях правильно подобранная структура может существенно ускорить решение. Моя задача — не сделать из вас участника соревнований по программированию, хотя это было бы хорошим бонусом, а научить работать со структурами данных и алгоритмами на примерах интересно поданных задач спортивного программирования. Пишите мне об успехах в своем обучении. Пишите также, если материалы этой книги заставят вас улыбнуться.

Онлайн-ресурсы

Примеры листингов, приведенные в книге, вы можете скачать с сайта издательства «Питер» по этому QR-коду:

Для кого эта книга

Книга предназначена для программистов, которые хотят научиться решать трудные задачи. Вы изучите множество структур данных и алгоритмов, узнаете об их преимуществах, видах задач, где эти алгоритмы и структуры применимы, и способах решения.

Весь код в книге написан на языке программирования Си. Но это не учебник по Си. Если у вас уже есть опыт работы с Си или C++, то смело окунайтесь в материал. Если же вы знакомы только с такими языками, как Java или Python, то я думаю,

что большую часть необходимого вы уясните по ходу чтения. Но все же желательно разобрать некоторые концепции Си заранее или при первой необходимости. В частности, я буду использовать указатели и динамическое выделение памяти. Поэтому, независимо от имеющегося опыта, вам может потребоваться освежить эти темы в памяти.

Лучшая, на мой взгляд, книга по работе с Си — это второе издание *C Programming: A Modern Approach* К. Н. Кинга (K. N. King). Даже если вы хорошо знакомы с Си, все равно рекомендую ее прочесть. Она оказывается отличным помощником, когда особенности этого языка вызывают сложности в понимании кода.

Язык программирования

Итак, я выбрал для этой книги именно Си, а не один из высокоуровневых языков вроде C++, Java или Python. Далее я вкратце опишу причину такого выбора, а также поясню пару других связанных с Си вещей.

Почему Си?

Основная причина выбора именно языка Си состоит в том, что я хотел рассказать вам о структурах данных и алгоритмах, начиная с азов. Когда нам понадобится хеш-таблица, мы будем создавать ее сами, не полагаясь на словари, хеш-карты или аналогичные структуры данных других языков. Когда нам неизвестна максимальная длина строки, мы создаем расширяемый массив, поскольку Си не станет выделять память за нас. Я хочу, чтобы вы полностью контролировали происходящее и за кадром ничего не оставалось. В этом-то мне и поможет Си.

Решение задач по программированию на Си служит хорошей основой в случае, если дальше вы захотите работать с C++. Если вы всерьез увлечетесь спортивным программированием, то стоит отметить, что C++ является самым популярным языком среди участников соревнований. Причина этого — его богатая стандартная библиотека и возможности генерировать код, ориентированный на высокую производительность.

Ключевое слово Static

В Си регулярные локальные переменные хранятся в так называемом *стеке вызовов*. При каждом вызове функции часть памяти стека задействуется для сохранения локальных переменных. Далее, когда функция делает возврат, эта память освобождается для последующего использования для других локальных переменных. Однако стек вызовов мал и не подходит для крупных массивов, которые будут встречаться в книге. Поэтому мы будем использовать ключевое слово `static`. Для локальной переменной

оно изменяет продолжительность хранения с динамической на статическую, в результате чего переменная сохраняет свое значение между вызовами функций. Побочным эффектом является то, что такие переменные *не* сохраняются в памяти вместе с регулярными локальными переменными, иначе при завершении функции их значения утрачивались бы. Они помещаются в отдельную область памяти, где им не приходится бороться за место под солнцем с другим содержимым стека вызовов.

При использовании ключевого слова `static` нужно иметь в виду, что такие переменные инициализируются всего один раз! В листинге 1 приводится краткий пример.

Листинг 1. Локальная переменная с ключевым словом static

```
int f(void) {
  static int x = 5; ❶
  printf("%d\n", x);
  x++;
}

int main(void) {
  f();
  f();
  f();
  return 0;
}
```

Я использовал `static` для локальной переменной `x` ❶. Без него все три раза выводом была бы `5`. Тем не менее благодаря `static` мы получаем:

```
5
6
7
```

Добавление файлов

С целью экономии места я не добавляю в листинги строки `#include`, которые необходимы для запуска программ Си. Все будет в порядке, если вы будете добавлять самостоятельно следующий код:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Освобождение памяти

В отличие от Java или Python, Си требует ручной настройки выделяемой памяти. Для выделения памяти служит функция `malloc`, а для освобождения — функция `free`.

Однако я не буду освобождать память по двум причинам. Во-первых, добавление соответствующих функций будет отвлекать от основной обучающей задачи кода.

Во-вторых, эти программы занимают место в памяти недолго: ваш код будет выполняться в нескольких тестовых примерах, и на этом все. Операционная система возвращает всю неосвобожденную память при закрытии программы, так что беспокоиться не о чем, даже если вы будете выполнять код по нескольку раз. Конечно же, невыполнение освобождения памяти на практике является безответственным подходом: никто не будет рад программе, которая чем дольше выполняется, тем больше потребляет памяти. Если вы хотите попрактиковаться в освобождении памяти, то можете сами добавлять в приводимые на страницах книги программы вызовы функции `free`.

Темы

Структуры данных и алгоритмы — слишком обширная область для полного описания в одной книге. Поэтому при отборе тем я опирался на три критерия.

Прежде всего, я выбирал темы с широкой областью применения, чтобы каждую из них можно было использовать при решении не только задач, подобных тем, что приведены в книге, но и многих других. В каждой главе рассматривается не менее двух задач. Первая обычно служит для того, чтобы представить структуру данных или алгоритм вместе с одним из классических случаев применения. Последующие задачи представляют дополнительные возможности рассматриваемой структуры данных или алгоритма. Например, в главе 5 изучается алгоритм Дейкстры. Если вы поищете информацию о нем в Google, то узнаете, что он используется для поиска кратчайших путей. И действительно, в первой задаче главы данный алгоритм применяется именно для этой цели. Однако во второй задаче мы подстраиваем его для поиска не одного, а нескольких кратчайших путей. Надеюсь, что по мере знакомства с каждой главой вы будете все больше узнавать о возможностях, ограничениях и тонкостях каждой техники решения.

Второй критерий — это простота реализации, чтобы не перегружать общий поток повествования. Я хотел, чтобы решение любой задачи укладывалось максимум в 150 строк кода, включая считывание входных данных, само решение и вывод. Поэтому структуры данных и алгоритмы, чья реализация занимает 200 или 300 строк, из практических соображений не рассматривались.

Наконец, я выбирал темы, доказательства корректности которых на мой взгляд являются убедительными и интуитивно понятными. Моя цель — научить вас конкретным структурам данных и алгоритмам, потому что вы наверняка читаете эту книгу, чтобы освоить эффективные подходы к решению задач. При этом я также надеюсь, что вам будет интересно узнать, *почему* та или иная техника работает. Поэтому я тайно преследовал еще одну цель: убедить вас, что рассматриваемые структуры данных или алгоритмы являются корректными. Формальных доказательств или чего-то подобного здесь не будет. Тем не менее если мой секретный

замысел сработает, то наряду с изучением самих структур данных и алгоритмов вы также убедитесь в их корректности. Не стоит ограничиваться чтением кода и восхищаться тем, как он всякий раз волшебным образом срабатывает. Магии здесь нет, и все принципы, заставляющие код работать, находятся в досягаемой для вашего понимания области.

Если у вас возникнет желание выйти за рамки материала книги, то я рекомендую заглянуть в приложение Б, где я добавил кое-какую дополнительную информацию к главам 1, 3, 4, 7 и 8.

Многим читателям будет полезно по ходу чтения книги заниматься дополнительной практикой и изучать сторонние материалы. В разделах «Примечания» в конце каждой главы приводятся ссылки на дополнительные ресурсы. Многие из них содержат дополнительные примеры кода и задачи. Помимо этого есть онлайн-ресурсы, которые содержат структурированный список задач и стратегий по их решению. Из всех известных мне ресурсов наиболее обширным в этом отношении является портал *Methods to Solve* братьев Стивена и Феликса Халимов: https://cpbook.net/methodstosolve.

Сайты с задачами

Каждая выбранная мной задача доступна на сайтах соревнований по программированию. Таких сайтов — множество, и на каждом из них содержатся сотни разных задач. Я старался, чтобы общее число источников было небольшим, но при этом достаточным для гибкого выбора наиболее подходящих задач. Каждый сайт потребует регистрации, и я советую сделать это сразу, чтобы потом не пришлось отвлекаться от решения предложенных в книге задач.

Вот список ресурсов, которые будут использоваться.

Название платформы	**Адрес**
Codeforces	codeforces.com
DMOJ	dmoj.ca
Kattis	open.kattis.com
POJ	poj.org
SPOJ	spoj.com
UVa	uva.onlinejudge.org

Описание каждой задачи начинается с указания ресурса, где ее можно найти, и номера/кода задачи.

Одни задачи написаны участниками сообществ, другие взяты из программ известных соревнований. Вот некоторые платформы, откуда были позаимствованы использованные в книге задания:

- Международная олимпиада по информатике (IOI) — престижное соревнование для учащихся старших классов школ. Каждую страну представляют по четыре человека, выступающих в индивидуальном зачете. Программа конкурса рассчитана на два дня, в течение которых участники решают множество задач по программированию.
- Канадский конкурс по программированию (CCC) и канадская олимпиада по программированию (CCO) — ежегодные соревнования для учащихся старших классов школ, организуемые Университетом Ватерлоо. CCC проходит в отдельных школах, после чего победители принимают участие в CCO в Университете Ватерлоо. Победители CCO представляют Канаду на IOI. Когда я был школьником, то неоднократно участвовал в CCC, но ни разу не попал на CCO — даже близко к этому не подошел.
- DWITE — онлайн-соревнование по программированию, проводившееся для подготовки студентов к участию в ежегодных конкурсах. К сожалению, DWITE больше не функционирует, но старые задачи — а они весьма хороши! — по-прежнему доступны.
- ACM East Central North America Regional Programming Contest — ежегодный конкурс для студентов университетов восточной части центрального региона Северной Америки, который проводится ассоциацией ACM. Победители приглашаются на финал международного студенческого соревнования по программированию (ACM International Collegiate Programming Contest, ICPC). В отличие от других упомянутых здесь конкурсов, на которых участники выступают в индивидуальном зачете, это соревнование является командным.
- SAPO — Южно-Африканская олимпиада по программированию. Ежегодное соревнование, которое состоит из трех раундов с растущей сложностью. По результатам конкурса отбираются участники, представляющие ЮАР на IOI.
- COCI — Хорватское открытое соревнование по информатике. Онлайн-конкурс, который проводится несколько раз в год. По его результатам формируется хорватская команда для участия в IOI.
- USACO — Олимпиада по программированию в США. Онлайн-соревнование, организуемое несколько раз в год. Самая сложная его часть — открытый чемпионат. В каждом соревновании участникам предлагаются четыре уровня сложности задач: бронзовый, серебряный, золотой и платиновый. По результатам отбирается команда для участия в IOI.

Полный список источников задач приведен в приложении Б. Когда вы отправляете код задачи, ресурс компилирует вашу программу и проверяет ее на тестовых примерах. Если она проходит все эти тесты за установленное время, то код принимается как верный. Принятые решения обозначают как `AC`. Если же ваша программа не проходит один или более тестов, то она не принимается. Такие программы обозначаются как `WA` (Wrong Answer). Третий вариант результата проверки относится к слишком медленным программам, которые обозначаются `TLE` (Time-Limit Exceeded). Обратите внимание, что `TLE` не означает корректность кода — если время истекает, то оставшиеся тесты не проводятся и возможны необнаруженные ошибки, ведущие к оценке `WA`.

На момент издания книги каждая из представленных в ней окончательных версий решений успешно проходила все тесты за установленное время. В рамках допустимого я стремился сделать код читаемым и отдавал предпочтение ясности, а не скорости, поскольку эта книга посвящена изучению структур данных и алгоритмов, а не выдавливанию максимальной производительности из программы, которая и без того справится со своей задачей.

Структура описания задачи

Прежде чем решать задачу, нужно четко определиться, что именно от нас требуется. Необходимо точно понимать саму задачу, а также предполагаемый способ считывания входных и генерации выходных данных. Поэтому каждая задача начинается с описания трех ее составляющих:

- **Условие.** Здесь я представляю контекст задачи и ее цель. Очень важно читать условие внимательно, чтобы в точности понимать, какую именно задачу вы решаете. Иногда невнимательное прочтение или неверная интерпретация даже, казалось бы, незначительных нюансов может вести к ошибочным решениям. Например, в одной из задач от нас потребуется купить «не менее» заданного количества яблок. Если же вместо этого вы будете покупать «ровно столько» яблок, то программа провалит некоторые из тестов.
- **Входные данные.** Автор задачи предоставляет тестовые примеры, которые необходимо выполнить, чтобы решение было зачтено правильным. Мы должны считывать из входных данных каждый тестовый пример, чтобы иметь возможность его обработать. Откуда нам знать, сколько там примеров? Какие данные находятся на каждой строке каждого примера? Если в них есть числа, то каков их диапазон? Если там строки, то каков предел их длины? Всю эту информацию я предоставляю в даннном разделе.
- **Выходные данные.** Может очень расстроить ситуация, в которой у нас будет программа, производящая верный ответ, но при этом проваливающая те-

стовые кейсы из-за вывода результата в неверном формате. Часть описания задачи, относящаяся к выводу, указывает, как именно он должен быть представлен. Например, в этом разделе будет описываться, сколько строк вывода должно получаться для каждого кейса, что размещать в каждой строке, нужно ли оставлять пустые строки до или после тестовых случаев и т. д. Кроме того, я устанавливаю для каждой задачи ограничение по времени: если программа не делает вывод для всех тестовых кейсов за этот установленный промежуток, она не проходит.

Я менял формулировки условий задач, не меняя их сути.

Для большинства задач книги мы будем считывать входные данные из стандартного ввода (stdin) и записывать результат в стандартный вывод (stdout) (только в двух задачах в главе 6 stdin и stdout не потребуются). Это означает, что нужно будет использовать такие функции Си, как `scanf`, `getchar`, `printf` и т. д. без явного открытия и закрытия файлов.

Задача. Очереди за продуктами

Разберем пример описания задачи. Я буду давать в скобках комментарии, указывая на наиболее важные моменты. Когда условие задачи станет понятным, мы ее решим. В отличие от других задач книги, это можно будет сделать с помощью конструкций и понятий, которые, надеюсь, вам уже знакомы. Если вы справитесь с этой задачей сами или пробежитесь по моему решению без особых затруднений, значит, вы вполне готовы к грядущим испытаниям. Если же у вас возникнут серьезные трудности, то могу порекомендовать для начала еще раз ознакомиться с основами программирования и/или порешать базовые задачки.

Начнем мы с задачи с платформы DMOJ под номером `lkp18c2p1` (найдите ее на сайте, чтобы после получения решения сразу отправить результат на проверку).

Условие

За продуктами выстроилось n очередей из известного количества людей. Далее по одному человеку будет прибывать еще m людей, занимая место в самой короткой очереди (то есть той, где меньше всего людей). Нужно определить количество людей в каждой очереди, к которой присоединяется каждый из m людей (внимательно обдумайте условие. Дальше идет пример, поэтому, если что-то остается неясным, попробуйте сопоставить его с описанием условия).

Предположим, что всего есть три очереди. В очереди 1 стоит три человека, в очереди 2 стоят двое, в очереди 3 — пятеро. Далее приходят еще четыре человека (прежде

чем читать далее, попробуйте понять, что на этом этапе будет происходить). Первый человек встает в очередь с двумя людьми, то есть очередь 2. Теперь в ней стоят трое. Второй прибывший присоединяется к очереди из трех людей, то есть к 1-й или 2-й, пусть будет 1-я. Теперь в очереди 1 стоят четыре человека. Третий пришедший присоединяется к очереди 2, где стоят трое, и ее размер увеличивается до 4. Последний подошедший встает в очередь с четырьмя людьми, выбирая между очередями 1 и 2. Пусть он выберет 1-ю, и в ней теперь будет пять человек.

Входные данные

Входные данные содержат один тестовый пример. В первой строке находятся два положительных целых числа: n — количество очередей; m — количество людей. Их максимальное значение — 100. Вторая строка содержит n положительных целых чисел, описывающих количество людей в каждой очереди до прибытия новых. Каждое из этих чисел не больше 100.

Вот пример входных данных:

```
3 4
3 2 5
```

(Обратите внимание, что здесь представлен всего один тестовый пример, следовательно, нужно считывать именно две строки входных данных.)

Выходные данные

Для каждого из m подошедших людей нужно вывести строку, содержащую количество людей в очереди, к которой они присоединяются.

Рабочий вывод для предложенного примера будет таким:

```
2
3
3
4
```

Время на решение тестового кейса ограничено тремя секундами (учитывая, что нужно обработать не более 100 новых людей для каждого случая, трех секунд вполне достаточно. Для решения не потребуются изощренные структуры данных или алгоритмы).

Решение

В задачах, где используются структуры данных, которые сложно создавать вручную, можно начинать со считывания входных данных. В других случаях я откладываю

этот код напоследок, поскольку обычно мы тестируем создаваемые функции путем их вызова с образцами значений. Нет нужды заниматься парсингом входных данных до момента, пока мы не будем готовы решить всю задачу.

Ключевые данные для решения — это количество людей в каждой очереди. Для их сохранения вполне подойдет массив, где каждая очередь будет занимать свой индекс. Этот массив я назову `lines`.

Каждый из прибывающих людей присоединяется к самой короткой очереди, поэтому нам потребуется вспомогательная функция, которая будет определять выбор очереди. Эта функция приведена в листинге 2.

Листинг 2. Индекс самой короткой очереди

```
int shortest_line_index(int lines[], int n) {
  int j;
  int shortest = 0;
  for (j = 1; j < n; j++)
    if (lines[j] < lines[shortest])
      shortest = j;
  return shortest;
}
```

Теперь, имея массив `lines`, а также значения `n` и `m`, можно решить тестовый пример. Соответствующий код дан в листинге 3.

Листинг 3. Решение задачи

```
void solve(int lines[], int n, int m) {
  int i, shortest;
  for (i = 0; i < m; i++) {
    shortest = shortest_line_index(lines, n);
    printf("%d\n", lines[shortest]);
    lines[shortest]++; ❶
  }
}
```

В каждой итерации внешнего цикла `for` мы вызываем вспомогательную функцию для извлечения индекса самой короткой очереди. Затем мы выводим длину этой очереди. Подошедший человек присоединяется именно к ней, поэтому ее размер надо увеличить на один ❶.

Далее осталось только считать входные данные и вызвать `solve`. Это выполняется в листинге 4.

Листинг 4. Функция main

```
#define MAX_LINES 100

int main(void) {
```

```
    int lines[MAX_LINES];
    int n, m, i;
    scanf("%d%d", &n, &m);
    for (i = 0; i < n; i++)
      scanf("%d", &lines[i]);
    solve(lines, n, m);
    return 0;
}
```

Объединение функций `shortest_line_index`, `solve` и `main` с добавлением в начале обязательных строк `#include` дает законченное решение, которое можно отправлять на проверку. При этом нужно правильно указывать язык программирования: для наших программ следует выбирать GCC, C99, C11 или иное обозначение компилятора для Си.

Если вы хотите протестировать код локально, прежде чем отправлять его на проверку, то это вполне можно сделать. Поскольку наши программы производят чтение из stdin, можно запустить программу и ввести тестовые данные вручную. Это приемлемо для небольших примеров, но будет утомительным, если процесс придется повторять либо требуется ввести большой объем данных. Более разумным вариантом будет сохранить входные данные в файле, а затем выполнить из командной строки *перенаправление ввода*, чтобы программа производила чтение из этого файла, а не с клавиатуры. Например, если сохранить тестовый пример для текущей задачи в `food.txt`, а скомпилированная программа будет называться `food`, то попробуйте следующее:

```
$ food < food.txt
```

Такой способ упрощает изменение тестового примера: просто отредактируйте содержимое `food.txt`, а затем запустите программу снова с перенаправлением ввода.

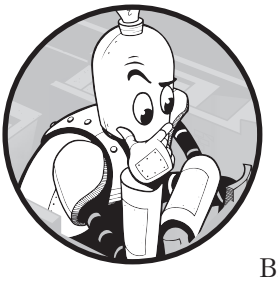
Поздравляю! Вы решили нашу первую задачу. Более того, теперь вы освоили структуру представления, которой будут соответствовать все последующие задачи книги.

Примечания

Задача «Очереди за продуктами» входила в программу конкурса LKP (Contest 2) на платформе DMOJ.

1

Хеш-таблицы



В этой главе мы решим две задачи, опираясь на средства эффективного поиска. В первой из них нужно будет проверить идентичность снежинок. Во второй мы будем определять, какие из слов являются сложносоставными. Вы увидите, что некоторые корректные подходы к решению оказываются недостаточно быстрыми. Для существенного повышения скорости вычислений мы используем структуру данных под названием «хеш-таблица», которую также изучим в деталях.

В завершение главы рассмотрим третью задачу: определим, сколькими способами можно удалить букву из слова, чтобы получить другое слово. В ней будут показаны риски необдуманного использования рассмотренной структуры данных, ведь при освоении чего-то нового всегда хочется сразу начать применять это везде!

Задача 1. Уникальные снежинки

Рассмотрим задачу под номером `cco07p2` с сайта DMOJ.

Условие

Дана коллекция снежинок, нужно определить, содержит ли она идентичные снежинки.

Каждая снежинка описывается шестью целыми числами, каждое из которых характеризует длину одного ее луча. Вот пример:

```
3, 9, 15, 2, 1, 10
```

При этом значения длин лучей могут повторяться, скажем, вот так:

```
8, 4, 8, 9, 2, 8
```

Но как понять, являются ли две снежинки идентичными? Рассмотрим несколько примеров.

Сначала сравним две снежинки:

```
1, 2, 3, 4, 5, 6
```

и

```
1, 2, 3, 4, 5, 6
```

Очевидно, что они идентичны, потому что числа одной совпадают с числами другой, находящимися в соответствующих позициях.

А вот второй пример:

```
1, 2, 3, 4, 5, 6
```

и

```
4, 5, 6, 1, 2, 3
```

Эти снежинки тоже идентичны, так как если начать с `1` во второй снежинке и продвигаться вправо, то сперва идут значения `1, 2, 3`, после чего происходит возврат в начало и последовательность продолжается значениями `4, 5, 6`. Оба этих сегмента вместе формируют снежинку, идентичную первой.

Каждую снежинку можно представить как круг. Тогда два предложенных здесь образца будут идентичными, потому что при правильном выборе начальной точки сопоставления во второй снежинке и следовании по часовой стрелке мы получаем первую снежинку.

Попробуем пример посложнее:

```
1, 2, 3, 4, 5, 6
```

и

```
3, 2, 1, 6, 5, 4
```

На основе только что рассмотренных вариантов можно сделать вывод, что эти снежинки не идентичны. Если начать с `1` во второй снежинке и продвигаться по кругу вправо до возвращения к точке отсчета, то мы получим `1, 6, 5, 4, 3, 2`. Эта форма далека от `1, 2, 3, 4, 5, 6` первой снежинки.

Тем не менее если начать с `1` во второй снежинке и перемещаться не вправо, а влево, тогда мы получим `1, 2, 3, 4, 5, 6`. Перемещение влево от `1` дает `1, 2, 3`, после чего происходит переход к правому краю и дальнейшее продвижение по значениям `4, 5, 6`.

Таким образом, две снежинки считаются идентичными и в случае, если числа совпадают при продвижении влево от точки отсчета.

Обобщая сказанное, можно заключить, что две снежинки идентичны, если они описываются одинаковыми последовательностями чисел либо если сходство обнаруживается при перемещении по этим последовательностям влево или вправо.

Входные данные

Первая строка входных данных является целым числом *n*, описывающим количество сравниваемых снежинок. Значение *n* будет находиться в диапазоне от 1 до 100 000. Каждая из следующих *n* строк характеризует одну снежинку: содержит шесть целых чисел, каждое из которых может иметь значение в диапазоне от 0 до 10 000 000.

Выходные данные

На выходе должна быть получена одна текстовая строка:

- Если идентичных снежинок не обнаружено, следует вывести:

  ```
  No two snowflakes are alike (нет одинаковых снежинок).
  ```

- Если есть хотя бы две идентичные снежинки, следует вывести:

  ```
  Twin snowflakes found (найдены одинаковые снежинки).
  ```

Время выполнения вычислений ограничено двумя секундами.

Упрощаем задачу

Одна из универсальных стратегий для решения задач по программированию состоит в проработке их упрощенных версий. Давайте немного разомнемся, снизив сложность нашей задачи.

Предположим, что мы работаем не со снежинками, состоящими из множества целых чисел, а с отдельными целыми числами. У нас есть коллекция, и нужно узнать, содержит ли она одинаковые числа. Проверить одинаковость двух целых чисел в Си можно с помощью оператора `==`. Мы будем тестировать все варианты пар чисел, и если найдем одинаковые числа хотя бы в одной паре, то остановимся и выведем:

```
Twin integers found (найдены одинаковые числа).
```

Если мы не обнаружим одинаковых чисел, то вывод будет таким:

```
No two integers are alike (нет одинаковых чисел).
```

Теперь создадим функцию `identify_identical` с двумя вложенными циклами для сравнения пар целых чисел, как показано в листинге 1.1.

Листинг 1.1. Поиск одинаковых целых чисел

```
void identify_identical(int values[], int n) {
  int i, j;
  for (i = 0; i < n; i++) {
    for (j = i+1; j < n; j++) {❶
      if (values[i] == values[j]) {
        printf("Twin integers found.\n");
        return;
      }
    }
  }
  printf("No two integers are alike.\n");
}
```

Мы передаем числа в функцию через массив `values`. Также мы передаем `n`, то есть количество чисел в массиве.

Обратите внимание, что мы начинаем внутренний цикл с `i + 1`, а не с `0` ❶. Если начать с `0`, то `j` будет равняться `i` и мы будем сравнивать элемент с самим собой, получая ложноположительный результат.

Протестируем `identify_identical` с помощью небольшой функции `main`:

```
int main(void) {
  int a[5] = {1, 2, 3, 1, 5};
  identify_identical(a, 5);
  return 0;
}
```

При выполнении кода вывод покажет, что функция определила совпадающую пару единиц. В дальнейшем я ограничусь минимумом тестов, но при этом важно, чтобы вы сами экспериментировали.

Решение основной задачи

Теперь попробуем изменить функцию `identify_identical` для решения задачи со снежинками. В код нужно будет внести два изменения.

1. Нужно работать не с одним, а с шестью целыми числами сразу. Для сравнения хорошо подойдет двумерный массив, в котором каждая строка будет снежинкой с шестью столбцами (по одному для каждого элемента).
2. Как мы выяснили, снежинки могут оказаться одинаковыми в трех случаях. К сожалению, это подразумевает, что для их сравнения уже не получится использовать `==`. Необходимо учесть возможности «перемещения вправо» и «перемещения влево» (не говоря уже о том, что `==` в Си для сравнения массивов не используется). Так что основным изменением имеющегося алгоритма станет правильное сопоставление снежинок.

Для начала напишем две вспомогательные функции: одну для проверки «перемещения вправо» и вторую для проверки «перемещения влево». Каждая из них будет получать параметры двух снежинок и стартовую точку обхода второй снежинки.

Проверка вправо

Заголовок функции для `identical_right`:

```
int identical_right(int snow1[], int snow2[], int start)
```

Чтобы определить, совпадают ли снежинки при «передвижении вправо», можно просканировать `snow1` с индекса `0` и `snow2` с индекса `start`. В случае обнаружения разногласия между сопоставляемыми элементами будет возвращаться `0`, указывая, что снежинки не идентичны. Если же все сопоставляемые элементы совпадут, возвращаться будет `1`. Таким образом, `0` означает `false`, а `1` — `true`.

В листинге 1.2 приводится первый вариант кода для этой функции.

Листинг 1.2. Определение идентичности снежинок перемещением вправо (с ошибкой!)

```
int identical_right(int snow1[], int snow2[],
                    int start) { //ошибка!
  int offset;
  for (offset =0; offset < 6; offset++) {
    if (snow1[offset] != snow2[start + offset]) ❶
      return 0;
  }
  return 1;
}
```