Spis treści

Przedmowa

Podręcznik *Wydajne programowanie w R* traktuje o zwiększaniu ilości pracy, jaką można wykonać w języku R w danym okresie czasu. Książka ta mówi zarówno o wydajności *obliczeniowej*, jak i wydajności *programisty*. Istnieje wiele doskonałych materiałów R omawiających takie tematy, jak wizualizacja (np. Chang 2012), przetwarzanie danych (np. Grolemund i Wickham 2016) czy rozwój pakietów (np. Wickham 2015). Jeszcze więcej jest materiałów, które pokazują, jak używać R w określonych dziedzinach, wliczając w to statystykę Bayesowską, uczenie maszynowe czy systemy informacji geograficznej. Niewiele jest jednak ujednoliconych materiałów na temat tego, jak sprawić, aby R działał w sposób efektywny. Porady, wskazówki i rozwijana przez lata wiedza na ten temat porozsiewane są po setkach stron internetowych, wątkach wiadomości e-mail oraz forach dyskusyjnych, co sprawia, że opanowanie sztuki tworzenia wydajnego kodu jest dla użytkowników R prawdziwym wyzwaniem.

Dzięki nauczaniu zrozumieliśmy, że problem ten dotyczy zarówno początkujących, jak i doświadczonych użytkowników. Bez względu na to, czy pytanie dotyczy korzystania z wektorów R w celu unikania pętli for, sposobów konfigurowania plików *.Rprofile* i *.Renviron*, czy też możliwości wykorzystania doskonałego interfejsu C++ języka R do realizacji cięższych zadań, pojęcie wydajności jest tutaj kluczowe. Celem tej książki jest przedstawienie wielu przydatnych wskazówek, ostrzeżeń i sztuczek w jednej spójnej formie, która przez wiele najbliższych lat stanowić będzie przydatne źródło wiedzy dla programistów R na dowolnym poziomie doświadczenia.

Treść tej książki odzwierciedla pytania, które nasi studenci z wielu różnych dyscyplin, poziomów umiejętności i gałęzi przemysłu zadawali nam na przestrzeni lat w celu przyspieszenia ich pracy z językiem R. Jak optymalnie skonfigurować mój system do pracy z językiem R? Jak stosować ogólne zasady informatyki (takie jak *nie powtarzaj się*; ang. *do not repeat yourself*, DRY) w skryptach R? Jak wykorzystać R w ramach wydajnego przepływu pracy, wliczając w to rozpoczęcie projektu, współpracę i udokumentowanie? I w końcu jak można szybko nauczyć się korzystać z nowych pakietów i funkcji?

Na te oraz wiele innych pytań podręcznik ten udziela odpowiedzi w ramach 10 niezależnych rozdziałów. Każdy rozdział rozpoczyna się od podstaw i w miarę postępu staje się coraz bardziej zaawansowany, tak więc każdy zawsze znajdzie w nich coś dla siebie. Choć bardziej zaawansowane tematy, takie jak programowanie równoległe i C++, mogą okazać się nieodpowiednie dla niedoświadczonych użytkowników R, książka ta pomaga przebrnąć przez trudy nauki języka R poprzez powolne nakreślanie zagadnień i osadzanie ich na solidnych fundamentach. Dzięki temu nawet doświadczeni użytkownicy R powinni

znaleźć tu nieznane im wcześniej porady. Podczas wykładania tego materiału często słyszymy „Dlaczego nikt wcześniej mi tego nie powiedział?”

Wydajne programowanie nie powinno być postrzegane jako dodatkowa opcja, zaś znaczenie wydajności wzrasta wraz z rozmiarem projektów i zbiorów danych. Właściwie pomysł na napisanie tej książki pojawił się w czasie nauczania kursu zatytułowanego *R for Big Data*, kiedy to szybko stało się jasne, że jeśli chce się pracować z dużymi zbiorami danych, nasz kod musi działać wydajnie. Nawet dla małych zbiorów danych wydajny kod, który jest szybki do napisania i szybki w działaniu, jest istotnym komponentem udanego projektu R. Okazało się, że pojęcie wydajnego programowania jest istotne dla wszystkich gałęzi społeczności R. Bez względu na to, czy jesteśmy sporadycznymi użytkownikami R (np. z powodu jego olbrzymiego zbioru pakietów statystycznych), chcemy stworzyć nowy pakiet, lub pracujemy z innymi nad dużym projektem, w którym wydajność ma charakter krytyczny, wydajność kodu będzie mieć znaczący wpływ na naszą produktywność.

Ostatecznie wydajność polega na uzyskiwaniu większych rezultatów przy mniejszym nakładzie pracy. Sprowadzając to do analogii samochodu, czy trasę o długości 1000 km wolałbyś przejechać na jednym tankowaniu (lub na pojedynczym ładowaniu baterii), czy może dotankowywać ciężki, niezgrabny i brzydki samochód co 50 km? A może wolałbyś wybrać całkiem inny, bardziej wydajny pojazd i cykl działania? Analogicznie, wydajny kod języka R jest lepszy od niewydajnego kodu pod każdym możliwym względem: jest łatwiejszy w czytaniu, pisaniu, uruchamianiu, udostępnianiu i utrzymywaniu. Książka ta nie jest w stanie dostarczyć wszystkich odpowiedzi na temat tego, jak produkować taki kod, ale z pewnością może ona zapewnić wiele pomysłów, przykładów kodu źródłowego i wskazówek, które pomogą rozpocząć podróż w odpowiednim kierunku.

Konwencje użyte w tej książce

W książce tej wykorzystywane są następujące konwencje typograficzne:

Kursywa
: Wskazuje nowe terminy, adresy URL, adresy e-mail, nazwy plików oraz rozszerzenia plików.

Pogrubienie
: Wskazuje nazwy pakietów R.

`Stała szerokość`
: Wykorzystywana w listingach programów, a także w obrębie akapitów przy odwoływaniu się do elementów programu, takich jak nazwy zmiennych lub funkcji, bazy danych, typy danych, zmienne środowiskowe, instrukcje i słowa kluczowe.

`Pogrubiona stała szerokość`
: Wskazuje polecenia lub inny tekst, który powinien zostać wprowadzony przez użytkownika.

Stała szerokość z kursywą
: Wskazuje tekst, który powinien zostać zastąpiony przez wartość podaną przez użytkownika lub nakreśloną przez kontekst. Również tekst komentarza w przykładach kodu.

Ten element symbolizuje wskazówkę lub sugestię.

Ten element symbolizuje uwagę ogólną.

Ten element wskazuje ostrzeżenie.

Korzystanie z przykładowego kodu

Materiał dodatkowy (przykładowy kod, ćwiczenia, itd.) dostępny jest do pobrania na stronie *https://github.com/csgillespie/efficient*.

Zadaniem tej książki jest pomóc Ci wykonać Twoją pracę. W ogólnym przypadku, jeśli do książki dołączony jest przykładowy kod, możesz wykorzystywać go w swoich programach i dokumentacji. Nie musisz kontaktować się z nami w celu uzyskania na to zgody, chyba że dokonujesz reprodukcji znacznej jego części. Przykładowo, napisanie programu, który wykorzystuje kilka fragmentów kodu pochodzącego z tej książki, nie wymaga uzyskania zgody. Sprzedaż lub dystrybucja przykładów z książek O'Reilly na płycie CD-ROM wymaga uzyskania zgody. Udzielanie odpowiedzi na pytanie poprzez zacytowanie przykładowego kodu z tej książki nie wymaga uzyskania zgody. Włączenie znaczącej ilości kodu z tej książki do dokumentacji Twojego produktu wymaga uzyskania zgody.

Nie wymagamy atrybucji, ale jesteśmy za nią wdzięczni. Atrybucja zawiera zazwyczaj tytuł, autora, wydawcę oraz numer ISBN. Na przykład: „*Wydajne programowanie w R* autorstwa Colina Gillespie i Robina Lovelace (O'Reilly/APN Promise). Copyright 2018 Colin Gillespie i Robin Lovelace, 978-83-7541-352-6".

Jeśli masz wątpliwości co do prawidłowego sposobu wykorzystywania przez Ciebie przykładów z tej książki, skontaktuj się z nami pod adresem *permissions@oreilly.com*.

O'Reilly Safari

Safari (dawniej Safari Books Online) jest opartą na członkostwie platformą szkoleniową dla przedsiębiorstw, instytucji rządowych, nauczycieli oraz osób indywidualnych.

Członkowie mają dostęp do tysięcy książek, filmów i ścieżek szkoleniowych, interaktywnych samouczków oraz starannie wyselekcjonowanych list odtwarzania od ponad 250 wydawców, do których należą między innymi O'Reilly Media, Harvard Business Review, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Adobe, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett oraz Course Technology.

Więcej informacji można znaleźć pod adresem *http://oreilly.com/safari*.

Jak się z nami skontaktować

Dla książki tej utworzyliśmy oddzielną witrynę, w ramach której publikować będziemy erratę, przykłady oraz wszelkie dodatkowe informacje. Strona ta dostępna jest pod adresem *http://bit.ly/efficient-r-programming*.

Dotyczące tej książki komentarze i pytania o charakterze technicznym prosimy wysyłać na adres *bookquestions@oreilly.com*.

Wszelkie aktualności oraz dodatkowe informacje na temat naszych książek, kursów i konferencji można znaleźć na stronie *http://www.oreilly.com*

Znajdź nas na Facebooku: *http://facebook.com/oreilly*

Obserwuj nas na Twitterze: *http://twitter.com/oreillymedia*

Oglądaj nas na YouTubie: *http://www.youtube.com/oreillymedia*

Podziękowania

Proces powstawania tej książki miał charakter otwarty, dzięki czemu wiele osób zgłosiło do niej poprawki mające na celu usunięcie drobnych błędów. Specjalne podziękowania należą się wydawnictwu O'Reilly za umożliwienie tego procesu, a także wszystkim tym, którzy zgłosili swoje zastrzeżenia poprzez GitHub: @Delvis, @richelbilderbeek, @adamryczkowski, @CSJCampbell, @tktan, @nachti, Conor Lawless, @timcdlucas, Dirk Eddelbuettel, @wolfganglederer, @HenrikBengtsson, @giocomai oraz @daattali.

Dziękujemy również recenzentom technicznym, Richardowi Cottonowi i Garrettowi Grolemundowi, za ich szczegółowe i niezwykle cenne uwagi.

Colin

Dla Esther, Nathana i Niamh. Dziękuję za Waszą cierpliwość.

Robin

Dziękuję moim współlokatorom w Cornerstone Housing Cooperative, że znosili moje antyspołeczne zachowanie w czasie pisania tej książki. Dziękuję również wszystkim w University of Leeds za zachęcanie mnie do realizacji projektów leżących poza zwyczajową pogonią za konferencjami i publikacjami akademickimi. Dziękuję też wszystkim osobom zaangażowanym w społeczność deweloperów open source, użytkownikom i przekazicielom informacji, dzięki którym wszystko to jest możliwe.

Rozdział 1

Wprowadzenie

Rozdział ten omawia szeroki zakres osób, dla których książka ta została napisana, zarówno pod względem ich doświadczenia z językiem R oraz samym programowaniem, jak i pod względem jej optymalnego wykorzystania. Każdy, kto bierze pod uwagę poprawę wydajności, powinien dokładnie wiedzieć, co oznacza ten termin. Temat ten w odniesieniu do wydajności algorytmicznej i wydajności programisty omówiony został w podrozdziale „Czym jest wydajność?" na stronie 4, zaś w odniesieniu do samego języka R – w podrozdziale „Czym jest wydajne programowanie w R?", również na tej samej stronie. Choć może się to wydawać oczywiste, to jednak mimo wszystko warto się zastanowić, dlaczego ktoś w ogóle miałby zawracać sobie głowę wydajnym kodem w czasach, gdy wyposażone w potężną moc obliczeniową komputery są tanie i ogólnodostępne? Temat ten poruszony został w podrozdziale „Dlaczego wydajność?" na stronie 7.

Na szczęście książka ta nie jest poświęcona wyłącznie językowi R. Rozwijane w czasie jej czytania umiejętności nie związane z językiem R, a wymagane do wydajnego programowania w R, omówione zostały w podrozdziale „Umiejętności uniwersalne zapewniające wydajność" na stronie 8. Choć jest to dość nietypowe jak na książkę o programowaniu, podrozdział ten wprowadza zagadnienia bezwzrokowego pisania i stosowania spójnego kodu, będące uniwersalnymi umiejętnościami, które powinny podnieść naszą wydajność również poza programowaniem. Jednak książka ta poświęcona jest przede wszystkim programowaniu, tak więc nie mogło w niej zabraknąć przykładowego kodu, jaki zamieszczony został w każdym rozdziale. Choć niniejszy rozdział otwierający ma bardziej charakter koncepcyjny i dyskursywny, to wcale nie jest on tu wyjątkiem: jego przedostatni podrozdział („Testy porównawcze i profilowanie" na stronie 10) opisuje dwa podstawowe narzędzia należące do zestawu programisty R, podając dla nich kilka ilustrujących przykładów. Na koniec należałoby wspomnieć kilka słów na temat tego, jak należy używać tej książki w połączeniu z powiązanymi z nią pakietami i kodem źródłowym. Zostało to omówione w podrozdziale „Materiały do książki" na stronie 15.

Wymagania wstępne

Jak zostało to podkreślone w następnym podrozdziale, w miarę czytania tej książki warto również wykonywać jej kod i samodzielnie z nim eksperymentować. Zamieszczona na początku każdego rozdziału część „Wymagania wstępne" pozwala upewnić się, że dysponujemy wszystkimi pakietami, jakie wymagane są w danym rozdziale. Wymagania wstępne dla tego rozdziału są następujące:

- Działająca instalacja języka R (patrz „Instalowanie R" na stronie 22).
- Zainstalowane i wczytane pakiety **microbenchmark**, **profvis** i **ggplot2** (zobacz „Instalowanie pakietów R" na stronie 24, aby poznać wskazówki dotyczące instalowania pakietów i utrzymywania ich w aktualnym stanie). Aby upewnić się, że pakiety te są zainstalowane, można wczytać je w następujący sposób:

```
library("microbenchmark")
library("profvis")
library("ggplot2")
```

Wymagania wstępne konieczne do wykonywania kodu zawartego w tej książce omówione zostały w podrozdziale „Materiały do książki" na stronie 15.

Dla kogo jest ta książka i jak z niej korzystać?

Podręcznik ten przeznaczony jest dla każdego, kto chce być w stanie szybciej wpisywać swój kod R, czyniąc go przy tym szybszym w wykonywaniu i bardziej skalowalnym. Rozważania na temat wydajności następują zazwyczaj *po* nauczeniu się podstaw języka R niezbędnych do analizowania danych. Mimo że książka ta może być również przydatna dla początkujących programistów, to jednak zakładamy, że czytelnik jest już obeznany z językiem R, lub też biegle posługuje się innym językiem programowania. Z tego powodu książka ta powinna być cennym nabytkiem dla osób o szerokim zakresie umiejętności, które należą do jednej z trzech poniższych grup:

Dla programistów z niewielkim doświadczeniem w R
: Książka ta pomaga zaznajomić się z udziwnieniami języka R, które pozwalają na jego wydajną pracę. Łatwo jest pisać w R powolny kod, jeśli będzie się go traktować podobnie jak inny język.

Dla użytkowników R z niewielkim doświadczeniem w programowaniu
: Książka ta prezentuje wiele koncepcji i sztuczek, częściowo zaczerpniętych z informatyki, które sprawiają, że praca staje się bardziej wydajna czasowo.

Dla początkujących użytkowników R z niewielkim doświadczeniem w programowaniu
: Książka ta od samego początku uczy poprawnych (lub przynajmniej tych mniej złych) nawyków. Złe nawyki łatwo jest przyswoić, ale dużo trudniej zgubić. Przeczytanie

tej książki już na początku kariery programisty może pomóc oszczędzić w przyszłości wiele godzin oszczędności na przeszukiwaniu sieci pod kątem omówionych w tej książce problemów.

Poprzez zidentyfikowanie grupy, do której należysz, będziesz w stanie w pełni wykorzystać potencjał tej książki. Zachęcamy do przeczytania *Wydajnego programowania w R* przy jednoczesnej realizacji własnego projektu R – czy to w formie zespołowego zadania w pracy, czy osobistego projektu w domu. Dlaczego? Książka ta ma szerszy zakres niż typowy podręcznik traktujący o programowaniu (przykładowo rozdział 4 omawia zagadnienia związane z zarządzaniem projektami), tak więc praca nad projektem wychodzącym poza ramy tej książki z pewnością pomoże przyswoić i zastosować w praktyce zawarte w niej koncepcje, zalecenia i przykłady. Taki rodzaj przejścia bezpośrednio od słów do czynów sprawi, że przyswojone informacje zostaną lepiej utrwalone: uczmy się poprzez pracę.

Do realizacji własnego projektu zachęcamy zwłaszcza tych, którzy są początkującymi użytkownikami języka R i wpisują się w tę ostatnią kategorię. Nie jest ważne, aby projekt ten doczekał się fazy ukończenia, gdyż ma on jedynie stanowić dodatkowy materiał do nauki R. Choć książka ta ma charakter ogólny, jest bardzo prawdopodobne, że język R wykorzystywany będzie przez nas w określonej dziedzinie. Z tego powodu zachęcamy do czytania tej książki w parze z materiałami szkoleniowymi przeznaczonymi dla danego obszaru wiedzy. Co więcej, zachęcamy, aby wszyscy czytelnicy wykorzystywali ten podręcznik obok innych materiałów do nauki języka R, takich jak liczne winiety, samouczki czy artykuły online (wspomniane w poniższej wskazówce), które wyprodukowane zostały przez społeczność języka R. Absolutnym minimum jest znajomość ramek danych, pętli i tworzenia prostych wykresów, o których można dowiedzieć się więcej ze wspomnianych materiałów.

Materiały do nauki R

Istnieje wiele miejsc zawierających materiały do nauki języka R – zarówno tych ogólnych, jak i tych przeznaczonych dla konkretnych dziedzin. Dla całkowitych nowicjuszy dostępnych jest kilka materiałów wprowadzających, jak choćby świetny podręcznik *Student's Guide to R* (*http://bit.ly/studentguider*), czy też nieco bardziej techniczny samouczek *http://bit.ly/icebreakR*.

Przewodniki oferowane są również w samym języku R. Dostęp do nich możesz uzyskać poprzez wpisanie w konsoli polecenia `help.start()`. Znajduje się tam chociażby oficjalny podręcznik *An Introduction to R*, który zawiera wiele cennych informacji, ale przez wielu postrzegany jest jako trudny. Wpisanie w konsoli polecenia `vignette()` spowoduje wyświetlenie listy przewodników, które spakowane są *wewnątrz Twojej instalacji R* (stąd też połączenie z Internetem nie jest wymagane). Aby podejrzeć winietę dla konkretnego tematu, wystarczy wprowadzić jej nazwę do uprzedniego polecenia. Na przykład, aby podejrzeć winietę wprowadzającą dla pakietu `dplyr`, należy skorzystać z polecenia `vignette(package = "dplyr", "introduction")`.

Kolejnym wczesnym przystankiem powinna być witryna *Comprehensive R Archive Network* (CRAN) (*https://cran.r-project.org/index.html*). Strona *Contributed Documentation* zawiera listę nadsyłanych materiałów (głównie samouczków) poświęconych wielu różnym tematom – od *tworzenia map* (*http://bit.ly/mapsinR*) aż po *ekonometrię* (*http://bit.ly/econometricR*). Nowa witryna *bookdown* (*https://bookdown.org/*) zawiera listę kompletnych (lub bliskich ukończenia) książek, które omawiają określone obszary wiedzy. Przykładem może tu być traktująca o pracy z danymi książka *R for Data Science* (*http://r4ds.had.co.nz/*) oraz podręcznik *Authoring Books with R Markdown* (*https://bookdown.org/yihui/bookdown/*) omawiający proces tworzenia książek za pomocą narzędzia R Markdown. Zachęcamy również do odwiedzania *blogosfery języka R* za pośrednictwem witryny *http://r-bloggers.com/*, korzystania z popularnych kanałów Twitter oraz przeglądania powiązanych z CRAN list mailingowych (*https://www.r-project.org/mail.html*), co pozwoli Ci uzyskać aktualne materiały, które mogą być wykorzystywane w połączeniu z tą książką.

Czym jest wydajność?

W życiu codziennym wydajność oznacza mniej więcej *poprawne działanie*. Wydajny pojazd jest w stanie pojechać daleko nie zużywając przy tym dużej ilości paliwa. Wydajny pracownik wykonuje swoją pracę szybko i bezstresowo. Z kolei wydajna żarówka świeci jasno przy minimalnym zużyciu energii. W tym ostatnim znaczeniu wydajność (η) zdefiniowana jest jako stosunek wykonanej pracy (W, generowanie światła) do nakładu pracy (Q, w tym przypadku zużycie energii):

$$\eta = \frac{W}{Q}$$

Jak przekłada się to na programowanie? Wydajny kod może być zdefiniowany ściśle lub ogólnikowo. Pierwszą, a przy tym bardziej ścisłą definicją wydajności jest *wydajność algorytmiczna*, która określa, jak szybko komputer jest w stanie wykonać część pracy przy wykorzystaniu danego fragmentu kodu. Koncepcja ta sięga samych początków obliczeń, o czym świadczy następujący cytat wyjęty z notatek Ady Lovelace (1842), jakie sporządziła ona na temat pracy Charlesa Babbage'a:

> Prawie dla wszystkich obliczeń możliwa jest wielka różnorodność układów następujących po sobie procesów, tak więc ich wybór na potrzeby silnika obliczeniowego musi być poparty różnymi rozważaniami. Podstawowym celem jest wybór takiego układu, który czas potrzebny do ukończenia wykonywanych obliczeń redukował będzie do minimum.

Drugą i zarazem szerszą definicją wydajnego przetwarzania jest *wydajność programisty*. Jest to ilość użytecznej pracy, jaką osoba (nie komputer) jest w stanie wykonać w danym czasie. Jest całkiem możliwe, że przepisanie naszego kodu w języku C sprawi, że stanie się on 100 razy szybszy. Jeśli jednak miałoby nam to zająć 100 osobogodzin, zabieg ten może nie być wart wymaganego wysiłku. Komputery mogą pracować dzień i noc. Ludzie niestety nie. Produktywność ludzka jest tematem rozdziału 4.

Po przeczytaniu tej książki powinieneś być w stanie pisać kod, który jest wydajny zarówno od strony *algorytmicznej*, jak i pod względem *produktywności*. Wydajny kod jest również zwięzły, elegancki i łatwy w utrzymaniu, co jest niezbędne podczas pracy z dużymi projektami. Nasuwa się jednak znacznie szersze pytanie: czym różni się wydajny kod w języku R od wydajnego kodu napisanego w dowolnym innym języku?

Czym jest wydajne programowanie w R?

Przedstawiony przez Adę problem dotyczący istnienia *wielu różnych sposobów* na rozwiązanie danego problemu jest kluczem do zrozumienia tego, jak wydajne programowanie w R różni się od wydajnego programowania w innych językach. R znany jest z tego, że pozwala użytkownikom rozwiązywać problemy na wiele sposobów. Wynika to z nieodłącznej elastyczności języka R, w ramach której prawie „wszystko może zostać zmodyfikowane po tym, jak zostanie utworzone" (Wickham 2014). Twórcy języka R, Ross Ihaka i Robert Gentleman, zaprojektowali go właśnie w ten sposób: już w ramach podstawowego pakietu R komórka w ramce danych może zostać wybrana na kilka różnych sposobów (z których trzy zostaną zilustrowane w podrozdziale „Przykład testu porównawczego" na stronie 11). Pozwala to programistom na korzystanie z języka zgodnie z ich preferencjami, ale może to dezorientować osoby poszukujące *właściwego* sposobu wykonywania rzeczy, obniżając przy tym efektywność, jeśli nie rozumiemy w pełni tego języka.

Rozgłos języka R za jego zdolność rozwiązywania problemów na wiele różnych sposobów wzrósł dodatkowo wraz z rozpowszechnieniem się pakietów rozwijanych przez społeczność. W tej książce skupimy się jedynie na najlepszych, patrząc z perspektywy wydajności, sposobach rozwiązywania problemów. Często poszukiwanie odpowiedzi na pytanie, dlaczego pewien sposób wykonywania rzeczy jest szybszy od innych, może być bardzo pouczające. Jeśli jednak naszym celem jest wyłącznie *wykonanie pracy*, musimy jedynie dowiedzieć się, jaki jest najbardziej efektywny sposób jej wykonania. W ten sposób wspomniana elastyczność języka R może stać się nieefektywna: choć znalezienie *pewnego* sposobu na rozwiązanie danego problemu może być łatwiejsze w R niż w przypadku innych języków, to jednak rozwiązywanie tego problemu w R może sprawić, że uzyskanie *najlepszego* rozwiązania będzie w nim trudniejsze, ponieważ oferuje on ich tak wiele. Autorzy tej książki bezpośrednio adresują ten problem poprzez prezentowanie w niej wyłącznie takich podejść, które są ich zdaniem najbardziej efektywne. Z jednej strony mają oni nadzieję, że zaufasz ich poglądom wypracowanym na przestrzeni wielu lat stosowania i nauczania języka R. Z drugiej strony liczą oni, że poddasz je w wątpliwość i przetestujesz

je z użyciem testów porównawczych, jeśli tylko podejrzewasz, że istnieje lepszy sposób na wykonanie jakiegoś zadania (a dzięki elastyczności języka R i jego zdolności do współpracy z innymi językami, faktycznie takowe mogą istnieć).

Powszechnie wiadomo, że w przypadku określonych zadań, w porównaniu do języków niskopoziomowych, kod R może odznaczać się niską wydajnością algorytmiczną, zwłaszcza jeśli został napisany przez kogoś, kto nie w pełni rozumie ten język. Warto jednak podkreślić liczne sposoby, w jakie R oferuje i zachęca do stosowania wydajności, a zwłaszcza wydajności programisty:

- Kod R nie jest kompilowany, ale wywołuje skompilowany kod. Oznacza to, że otrzymujemy korzyści płynące z obu tych światów: R nie wymaga uciążliwego kompilowania kodu przed uruchomieniem, a przy tym oferuje znaczące wzrosty szybkości uzyskiwane przez wywoływanie za kulisami skompilowanego kodu napisanego w językach C, FORTRAN oraz innych językach programowania.
- R jest językiem funkcyjnym i obiektowym (Wickham 2014). Oznacza to, że można w nim pisać złożone i elastyczne funkcje, które w ramach pojedynczej linii kodu pozwalają na wykonywanie olbrzymiej ilości pracy.
- R jako swoją pamięć wykorzystuje pamięć RAM. Może się to wydawać oczywiste, ale warto przypomnieć, że pamięć ta jest szybsza niż jakikolwiek system dyskowy. Tym samym w porównaniu do baz danych, R jest bardzo szybki przy wykonywaniu operacji manipulowania danymi, przetwarzania i modelowania. Pamięć RAM jest teraz tańsza niż kiedykolwiek, przez co ewentualne wady tej funkcjonalności są jeszcze mniejsze niż do tej pory.
- R wspierany jest przez wiele doskonałych zintegrowanych środowisk programowania. Środowisko, w którym programujemy, może mieć olbrzymi wpływ na wydajność programisty, ponieważ może ono szybko dostarczyć mu pomoc, pozwolić na interaktywne sporządzanie wykresów i umożliwić mu ścisłą integrację projektu R z innymi aspektami realizacji projektu, takimi jak zarządzanie plikami, wersjami kodu czy systemami interaktywnej wizualizacji, jak zostało to omówione w podrozdziale „RStudio" na stronie 37.
- R cechuje duża i aktywna społeczność użytkowników. Znacząco podnosi to wydajność, ponieważ jeśli napotkamy problem, którego nie jesteśmy w stanie sami rozwiązać, możemy najzwyczajniej zapytać o niego inną osobę. Jeśli jest to nowe, klarowne i możliwe do odtworzenia pytanie, które zadaliśmy na forum *Stack Overflow* lub na odpowiedniej liście dyskusyjnej R[1], zapewne w ciągu kilku minut uzyskamy odpowiedź od doświadczonego programisty R. Oczywistą zaletą tego rodzaju systemu wsparcia jest to, że korzyści płynące z takiej odpowiedzi będą od tego momentu dostępne dla każdego.

1 *https://www.r-project.org/mail.html*

Wydajne programowanie w R jest implementacją wydajnych praktyk programowania w R. Każdy język jest inny, przez co wydajny kod w R niekoniecznie musi przypominać wydajny kod w innym języku. Wiele pakietów zostało zoptymalizowanych pod kątem wydajności, tak więc dla określonych operacji osiągnięcie maksymalnej wydajności obliczeniowej może sprowadzać się do zwykłego wyboru właściwego pakietu i jego poprawnego wykorzystywania. W języku R ten sam wynik można uzyskać na wiele różnych sposobów, przy czym część z nich może wykonywać się bardzo powoli. Z tego powodu *niepisanie* powolnego kodu powinno mieć wyższy priorytet niż pisanie szybkiego kodu.

Powracając do nakreślonej w przedmowie analogii o dwóch samochodach, wydajne programowanie w R dla pewnych przypadków użycia może po prostu oznaczać zamianę naszej starej, ciężkiej i paliwożernej funkcji SUV na lekki velomobil. Poszukiwanie optymalnej wydajności często przynosi malejące zyski, dlatego też ważne jest, aby wyszukać w naszym kodzie wąskie gardła i w ten sposób zapewnić maksymalne wzrosty w wydajności obliczeniowej. Wracając jeszcze do popularności R jako elastycznego języka, wydajne programowanie w R można interpretować jako poszukiwanie rozwiązania, które jest wystarczająco szybkie w kontekście wydajności obliczeniowej, ale równie szybkie w kontekście wydajności programisty. W końcu zarówno my, jak i nasi współpracownicy, na pewno mamy ciekawsze i ważniejsze rzeczy do zrobienia poza pracą, tak więc ważne jest, aby wykonać tę pracę szybko i przeznaczyć ten czas na inne interesujące nas zajęcia.

Dlaczego wydajność?

Komputery stają się coraz potężniejsze. Czy nie sprawia to, że wydajne przetwarzanie jest coraz mniej potrzebne? Odpowiedź jest prosta: nie. W erze Big Data i ulegających coraz większej stagnacji szybkościom zegarów komputerowych (patrz rozdział 8) obliczeniowe wąskie gardła mogą teraz ograniczać naszą pracę w stopniu większym niż kiedykolwiek. Wydajny programista może „rozwiązywać bardziej złożone zadania, zadawać bardziej ambitne pytania i uwzględniać w swoich badaniach bardziej zaawansowane analizy" (Visser i in. 2015).

Poniższy przykład ilustruje znaczenie wydajności w sytuacjach krytycznych. W ramach kontraktu zawartego z brytyjskim Departamentem Transportu, Robin pracował nad budową narzędzia Propensity to Cycle Tool, będącego aplikacją online, która miała być gotowa do krajowego wdrożenia w czasie krótszym niż cztery miesiące. Dla tego konkretnego projektu stworzył on funkcję `line2route()` z pakietu stplanr, mającą na celu generowanie tras za pomocą API *CycleStreets*[2]. Potrzebnych było setki tysięcy tras, ale ku jego przerażeniu kod niemal całkowicie zwolnił już po kilku tysiącach tras. Zawarty kontrakt był zagrożony. Po wyeliminowaniu innych problemów i wyprofilowaniu kodu (o czym traktuje podrozdział „Profilowanie kodu" na stronie 141) okazało się, że przyczyną spowolnienia był błąd w funkcji `line2route()`. Mówiąc dokładniej, pojawił się w niej *problem rosnących wektorów*, omówiony w podrozdziale „Przydział pamięci" na stronie 53.

2 *http://www.cyclestreets.net/*

Rozwiązanie było proste. *Pojedyncze zatwierdzenie*[3] modyfikacji kodu spowodowało, że funkcja `line2route()` stała się ponad dziesięć razy szybsza i znacząco krótsza. Modyfikacja ta pozwoliła ocalić cały projekt przed porażką. Morał z tej historii jest taki, że wydajne programowanie nie jest wyłącznie pożądaną umiejętnością, lecz może niekiedy okazać się niezbędne.

Wiele koncepcji i umiejętności jest agnostycznych językowo. Duża część wiedzy z tego podręcznika powinna mieć również zastosowanie przy programowaniu w innych językach (jak również przy wykonywaniu innych czynności technicznych poza programowaniem). Jednak w przypadku jednego konkretnego języka powinniśmy wyraźnie skupić się na wydajności. W języku R nawet zwyczajna podmiana funkcji na funkcję z innego pakietu może znacząco podnieść wydajność, o czym możemy przekonać się na przykładzie wczytywania plików tekstowych w rozdziale 5. Taki poziom szczegółowości, wraz z możliwymi do odtworzenia przykładami, nie byłby możliwy w książce poświęconej ogólnym zagadnieniom wydajnego programowania. Umiejętności zapewniające wydajną pracę, które mają zastosowanie również poza programowaniem w R, omówione zostały w kolejnym podrozdziale.

Umiejętności uniwersalne zapewniające wydajność

Znaczenie *wydajnego kodu R*, w przeciwieństwie do generycznego *wydajnego kodu*, powinno być już zrozumiałe po przeczytaniu dwóch poprzednich podrozdziałów. Nie oznacza to jednak, że umiejętności i koncepcje omówione w tej książce nie mogą być przenoszone do innych języków programowania czy nawet zadań niezwiązanych z programowaniem. Rozwój tych uniwersalnych umiejętności usprawni nie tylko nasze programowanie w R, ale również inne aspekty naszego życia zawodowego. Dwie spośród tych umiejętności są szczególnie ważne: pisanie bezwzrokowe oraz posługiwanie się spójnym stylem.

Pisanie bezwzrokowe

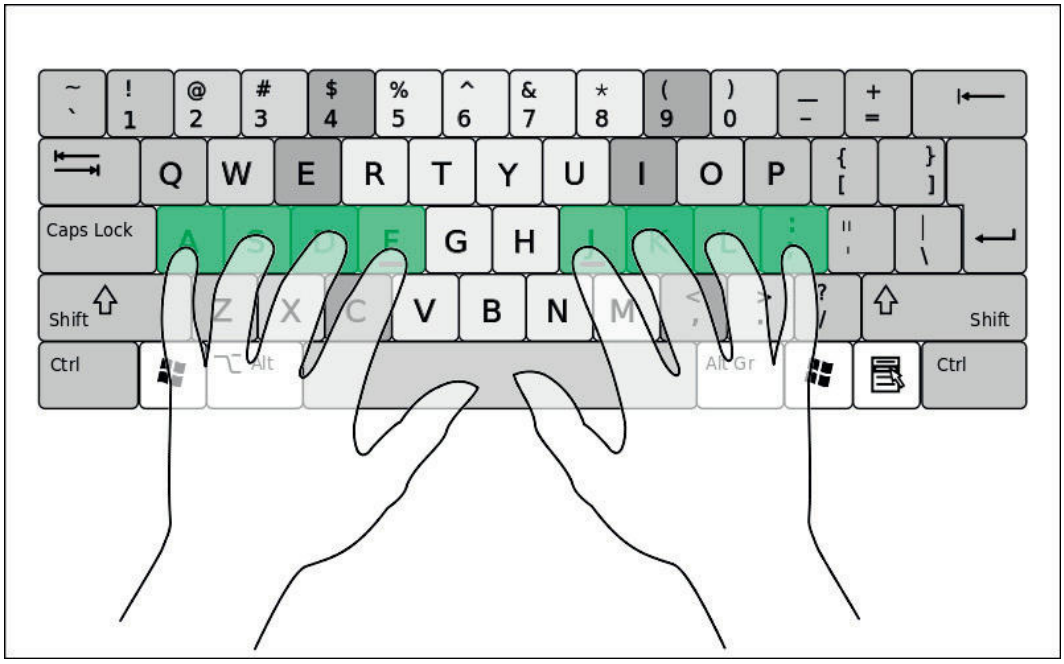
Drugą stroną medalu wydajności jest wydajność programisty. Istnieje wiele rzeczy, które są w stanie zwiększyć produktywność naszą i naszych współpracowników, jak choćby stosowanie się do zasady Philippa Janerta „myśl więcej, rób mniej" (Janert 2010). Badania pokazują, że dobra dieta, aktywność fizyczna, odpowiednia ilość snu oraz zdrowy balans między życiem i pracą mogą zwiększyć naszą szybkość i efektywność w pracy (Jensen 2011; Pereira i in. 2015; Grant, Spurgeon i Wallace 2013).

Zachęcamy do samodzielnego przemyślenia tych badań, gdyż nie jest to książka o samopomocy, lecz podręcznik traktujący o programowaniu. Istnieje jednak jedna nieprogramistyczna umiejętność, która *może* mieć olbrzymi wpływ na naszą produktywność: pisanie bezwzrokowe. Opanowanie tej umiejętności jest w dużej mierze proste, a pozwoli nam ona szybko pisać, modyfikować i testować nasz kod R. Poprawne opanowanie sztuki

3 *http://bit.ly/refactorline2route*

bezwzrokowego pisania będzie nam się wielokrotnie odpłacać w dalszym okresie naszego programistycznego życia (oczywiście uzyskane w ten sposób korzyści nie są ograniczone wyłącznie do programowania w R).

Podstawową różnicą pomiędzy osobą piszącą bezwzrokowo a kimś, kto nieustannie spogląda na klawisze (lub kto używa do pisania jedynie dwóch lub trzech palców), jest rozmieszczenie rąk na klawiaturze. Pisanie bezwzrokowe polega na takim rozmieszczeniu rąk nad klawiaturą, że każdy palec u obu rąk dotyka lub znajduje się bezpośrednio nad konkretną literą (rysunek 1-1). Opanowanie tej umiejętności wymaga czasu i pewnej dyscypliny. Na szczęście istnieje wiele materiałów, które pomogą nam szybko i skutecznie wyrobić ten nawyk, wliczając w to chociażby projekty open source *Klavaro*[4] i *TypeFaster*[5].



Rysunek 1-1 *Pozycja wyjściowa do pisania bezwzrokowego, w której palce rozmieszczone są nad początkowymi klawiszami. Źródło: Wikipedia na licencji Creative Commons.*

Spójny styl i konwencje kodowania

Wypracowanie nawyku stosowania przejrzystego i spójnego stylu podczas pisania czegokolwiek – kodu bądź poezji – przynosić nam będzie korzyści w wielu innych projektach, zarówno tych programistycznych, jak i tych, które z programowaniem nie mają nic wspólnego. Jak zostało to nakreślone w podrozdziale „Styl kodowania” na stronie 178, styl jest w pewnym zakresie osobistą preferencją programisty. Na tym etapie zachęcamy do zapoznania się ze stosowanymi w tej książce konwencjami, co pozwoli zmaksymalizować jej czytelność.

4 *https://sourceforge.net/projects/klavaro/*
5 *https://sourceforge.net/projects/typefaster/*

W książce tej posługujemy się spójnym zestawem konwencji w celu odwoływania się do kodu.

- Nazwy pakietów prezentowane są czcionką pogrubioną, np. **dplyr**.
- Nazwy funkcji prezentowane są czcionką dla kodu, zaś po nich następuje para nawiasów, np. `plot()` lub `median()`.
- Pozostałe obiekty R, takie jak dane lub argumenty funkcji, oznaczone są czcionką dla kodu bez nawiasów, np. `x` lub `name`.
- Okazjonalnie podawać będziemy nazwę pakietu, z którego pochodzi dana funkcja, używając symbolu podwójnego dwukropka np. `microbenchmark::microbenchmark()`. Zwróćmy uwagę, że stosowanie tej notacji można uznać za wydajne tylko wtedy, gdy dana funkcja z pakietu potrzebna nam jest tylko raz, jako że nie musimy wówczas załączać całego pakietu.

Pojęcia testów porównawczych i profilowania kodu nie są charakterystyczne wyłącznie dla języka R. W języku tym są one jednak stosowane w sposób szczególny, co zostało zaprezentowane w kolejnym podrozdziale.

Testy porównawcze i profilowanie

Wykonywanie testów porównawczych (ang. *benchmarking*) i profilowanie kodu (ang. *profiling*) jest kluczem do wydajnego programowania, zwłaszcza w języku R. Testy porównawcze polegają na wielokrotnym testowaniu wydajności konkretnych operacji. Profilowanie polega natomiast na uruchamianiu wielu linii kodu w celu znalezienia w kodzie wąskich gardeł. Do pełnego zrozumienia wydajności niezbędne są obie te metody, dlatego też będziemy z nich korzystać w tej książce. Ich fundamentalne znaczenie dla praktyki wydajnego programowania powoduje, że muszą one zostać omówione już w tym wprowadzającym rozdziale, mimo że przez wielu są one uznawane za temat średniozaawansowany lub zaawansowany.

Testy porównawcze można w pewnym sensie postrzegać jako elementy składowe profilów. Profilowanie można rozumieć jako automatyczne wykonywanie wielu testów porównawczych dla każdego wiersza w skrypcie, a następnie porównywanie uzyskanych wyników wiersz po wierszu. Ponieważ testy porównawcze są mniejsze, łatwiejsze i bardziej modułowe, omówimy je jako pierwsze.

Wykonywanie testów porównawczych

Modyfikowanie elementów pomiędzy jednym testem porównawczym a drugim, wraz z rejestrowaniem wyników uzyskiwanych po tych zmianach, pozwala nam określić najszybszy fragment kodu. Testy porównawcze są ważną częścią zestawu narzędzi wydajnego programisty: może nam się *wydawać*, że nasz kod jest szybszy od kodu innej osoby, ale to testy porównawcze pozwalają nam to *udowodnić*. Najprostszym sposobem przetestowania

wybranego fragmentu kodu jest skorzystanie z funkcji `system.time()`. Należy jednak pamiętać, że pobieramy tutaj próbę. Nie oczekiwalibyśmy przecież, że pojedyncza osoba w Londynie będzie reprezentatywna dla całej populacji Wielkiej Brytanii. Podobnie jest w przypadku testu porównawczego, gdzie pojedynczy test dostarcza nam pojedynczą obserwację na temat zachowania naszej funkcji. Z tego powodu pomiar czasu musimy powtórzyć wiele razy w ramach pętli.

Alternatywnym sposobem wykonywania testów porównawczych jest skorzystanie z elastycznego pakietu microbenchmark. Pakiet ten pozwala nam w łatwy sposób wielokrotnie uruchamiać każdą z funkcji (domyślnie 100 razy), co pozwala wykryć różnice w wydajności kodu na poziomie mikrosekund. Po zakończeniu testu otrzymujemy wygodne podsumowanie uzyskanych wyników: minimum/maksimum, kwartyl dolny/górny oraz średnia/mediana. Warto skupić się na środkowej wartości czasu, by oszacować standardowy czas wykonywania, a także na kwartylach, które pomogą nam zrozumieć zmienność czasu wykonywania.

Przykład testu porównawczego

Dobrym przykładem może być testowanie różnych metod do wyszukiwania pojedynczej wartości w ramce danych. Zwróćmy uwagę, że każdy argument w poniższym teście jest wyrażeniem poddawanym ewaluacji (w przypadku testów złożonych z wielu linii kodu wyrażenie do ewaluacji może zostać ujęte w nawiasy klamrowe {}).

```
library("microbenchmark")
df = data.frame(v = 1:4, name = letters[1:4])
microbenchmark(df[3, 2], df[3, "name"], df$name[3])
# Unit: microseconds
#          expr     min    lq  mean median    uq   max neval cld
#      df[3, 2]   17.99 18.96 20.16  19.38 19.77 35.14   100   b
# df[3, "name"]   17.97 19.13 21.45  19.64 20.15 74.00   100   b
#    df$name[3]   12.48 13.81 15.81  14.48 15.14 67.24   100   a
```

Wyniki testu podsumowują, ile czasu zajęło wykonanie każdego zapytania: dla każdej liczby ewaluacji (`neval`, w tym przypadku z domyślną wartością 100) podawane są wartości minimum (`min`), dolny i górny kwartyl (odpowiednio `lq` i `uq`) oraz średnia (`mean`), mediana (`median`) i maksimum (`max`). Kolumna `cld` ukazuje względną rangę każdego wiersza w formacie *compact letter display*: w tym przypadku `df$name[3]` wypada najlepiej, z rangą `a` oraz średnim czasem około 25% krótszym w porównaniu do dwóch pozostałych funkcji.

W przypadku korzystania z funkcji `microbenchmark()` należy zwrócić szczególną uwagę na jednostki. W poprzednim przykładzie każde wywołanie funkcji zajmuje około 20 *mikrosekund*, co sugeruje, że w czasie jednej sekundy może zostać wykonanych około 50 000 wywołań funkcji. W przypadku porównywania ze sobą szybkich funkcji standardowymi jednostkami będą: