



ALGORYTMY  
BEZ TAJEMNIC

THOMAS H. CORMEN

Tytuł oryginału: Algorithms Unlocked

Tłumaczenie: Zdzisław Płoski

ISBN: 978-83-283-6736-4

© Helion 2013, 2018, 2020  
All rights reserved.

© 2013 Massachusetts Institute of Technology

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Helion SA dołożyło wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie bierze jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Wydawnictwo HELION nie ponosi również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion SA  
ul. Kościuszki 1c, 44-100 GLIWICE  
tel. 32 231 22 19, 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!  
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres  
<http://helion.pl/user/opinie/algbvv>  
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

---

# Spis treści

<b>Przedmowa</b>	<b>9</b>
<b>1 Co to są algorytmy i dlaczego warto poświęcać im uwagę?</b>	<b>15</b>
Poprawność	16
Użytkowanie zasobów	17
Algorytmy komputerowe dla niekomputerowców	19
Algorytmy komputerowe dla komputerowców	20
Co czytać dalej	21
<b>2 Jak opisywać i oceniać algorytmy komputerowe</b>	<b>23</b>
Jak opisywać algorytmy komputerowe	23
Jak charakteryzować czasy działania	29
Niezmienniki pętli	33
Rekursja	34
Co czytać dalej	36
<b>3 Algorytmy sortowania i wyszukiwania</b>	<b>37</b>
Wyszukiwanie binarne	39
Sortowanie przez wybieranie	43
Sortowanie przez wstawianie	46
Sortowanie przez scalanie	50
Sortowanie szybkie	59
Podsumowanie	66
Co czytać dalej	69
<b>4 Dolne ograniczenie sortowania i sposoby jego przewyciężenia</b>	<b>71</b>
Reguły sortowania	71
Dolne ograniczenie sortowania przez porównania	72
Pokonywanie ograniczenia dolnego w sortowaniu przez zliczanie	73
Sortowanie pozycyjne	79
Co czytać dalej	81

<b>5 Skierowane grafy acykliczne</b>	<b>83</b>
Skierowane grafy acykliczne	87
Sortowanie topologiczne	87
Jak reprezentować graf skierowany	90
Czas działania sortowania topologicznego	92
Ścieżka krytyczna w diagramie PERT	92
Najkrótsza ścieżka w skierowanym grafie acyklicznym	96
Co czytać dalej	100
<b>6 Najkrótsze ścieżki</b>	<b>101</b>
Algorytm Dijkstry	102
Algorytm Bellmana-Forda	111
Algorytm Floyda-Warshalla	115
Co czytać dalej	123
<b>7 Algorytmy napisowe</b>	<b>125</b>
Najdłuższy wspólny podciąg	125
Zamiana napisu na inny	130
Dopasowywanie napisów	137
Co czytać dalej	144
<b>8 Podstawy kryptografii</b>	<b>145</b>
Proste szyfry podstawieniowe	146
Kryptografia z kluczem symetrycznym	147
Kryptografia z kluczem jawnym	151
Kryptosystem RSA	153
Kryptosystemy hybrydowe	160
Obliczanie liczb losowych	161
Co czytać dalej	162
<b>9 Kompresja danych</b>	<b>163</b>
Kody Huffmana	164
Faksy	170
Kompresja LZW	171
Co czytać dalej	180

<b>10 Trudne (?) problemy</b>	<b>181</b>
Brązowe furgonetki	181
Klasy P i NP oraz NP-zupełność	184
Problemy decyzyjne i redukcje	186
Problem matka	189
Próbnik problemów NP-zupełnych	191
Ogólne strategie	204
Perspektywy	206
Problemy nierozstrzygalne	208
Podsumowanie	210
Co czytać dalej	211
<b>Literatura</b>	<b>213</b>
<b>Skorowidz</b>	<b>215</b>



---

# 1 Co to są algorytmy i dlaczego warto poświęcać im uwagę?

Zacznijmy od pytania, które mi często zadają: „Co to jest algorytm?”<sup>1</sup>.

Ogólna odpowiedź mogłaby być taka: „zbiór kroków prowadzących do wykonania zadania”. Na co dzień stosujesz różne algorytmy. Masz algorytm mycia zębów: otwierasz tubkę z pastą, bierzesz szczoteczkę do ręki, wyciskasz na szczoteczkę tyle pasty, ile trzeba, zamykasz tubkę, wkładasz szczoteczkę do jednej ćwiartki paszczy, przesuwasz nią w górę i w dół (oraz w prawo i w lewo) przez  $N$  sekund itd. Jeśli musisz dojeżdżać do pracy, masz swój algorytm dojazdu do pracy. I tak dalej.

Jednak ta książka zajmuje się algorytmami, które działają na komputerach, a ogólnie rzecz ujmując — na urządzeniach obliczeniowych. Z algorytmami wykonywanymi na komputerach jest podobnie jak z tymi, które wykonujesz na co dzień. Korzystasz z GPS-u, żeby znaleźć drogę? Żeby ją odnaleźć, wykonuje on algorytm, który nazywamy „najkrótsza ścieżka”. Kupujesz coś w Internecie? Używasz wtedy (a przynajmniej należałoby!) bezpiecznej witryny sieciowej, która wykonuje algorytm szyfrowania. Gdy kupujesz jakieś towary w Sieci, kto Ci je dostarcza? Firma kurierska? Korzysta ona z algorytmów przydzielających paczki do furgonetek, a potem ustalających kolejność, w jakiej każdy kierowca powinien te paczki rozwozić. Algorytmy działają w komputerach na każdym kroku: w Twoim laptopie, smartfonie, na serwerach lub w systemach wbudowanych (np. w Twoim samochodzie czy w mikrofalówce, w systemach klimatyzacji) — wszędzie!

Co różni algorytmy wykonywane na komputerach od algorytmu wykonywanego przez Ciebie? Ty potrafisz tolerować algorytm, jeśli jest on niedokładnie opisany, a komputer nie. Na przykład, jeśli jedziesz do pracy, Twój algorytm jedź-do-pracy mógłby zawierać taką klauzulę: „jeśli jest tłoczno, wybierz inną trasę”. Ty możesz wiedzieć, co rozumiesz pod określeniem „jest tłoczno”, natomiast komputer tego nie pojmie.

Dlatego algorytm komputerowy jest zbiorem kroków prowadzących do wykonania zadania, opisanym na tyle precyzyjnie, że potrafi go wykonać komputer. Na czym polega ta precyzja, wiesz, jeśli masz choć trochę doświadczenia w programowaniu komputerów w Javie, C, C++, Pythonie, Fortranie, Mathlabie lub w czymś podobnym. Jeśli nie masz w dorobku ani jednego programu komputerowego, to niewykluczone, że poczujesz ten stopień dokładności, przyglądając się, jak opisują algorytmy w tej książce.

Przejdźmy do następnego pytania: „Czego oczekujemy od algorytmu komputerowego?”.

---

<sup>1</sup> Lub, jak by rzekł mój kolega, z którym gram w hokeja: „What’s a nalgorithm?”.

Algorytmy komputerowe rozwiązują problemy obliczeniowe. Od algorytmu komputerowego oczekujemy dwóch rzeczy: mając dane do problemu, powinien zawsze wytwarzać poprawne rozwiązanie tego problemu, a robiąc to, powinien oszczędnie zużywać zasoby obliczeniowe. Przyjrzyjmy się obu tym postulatam.

## Poprawność

Co oznacza poprawne rozwiązanie problemu? Na ogół potrafimy precyzyjnie określić, co powinno zawierać poprawne rozwiązanie. Na przykład, jeśli Twój GPS wytwarza poprawne rozwiązanie, gdy chodzi o znalezienie najlepszej trasy podróży, to mogłoby to polegać na wybraniu takiej trasy spośród wszystkich możliwych do wytyczenia między miejscem Twojego pobytu a miejscem docelowym, którą dotrzesz tam najszybciej. Albo trasy możliwie najkrótszej. Albo takiej, która nie tylko poprowadzi Cię do celu najszybciej, lecz również pozwoli uniknąć płacenia myta. Jasne, że informacje, których Twój GPS używa do wyznaczenia trasy, mogą nie odpowiadać rzeczywistości. O ile nie ma on dostępu w czasie rzeczywistym do danych o ruchu drogowym, mógłby przyjąć, że czas potrzebny do przebycia drogi równa się długości drogi podzielonej przez dopuszczalną na niej prędkość. Jeśli jednak droga jest zatłoczona, GPS mógłby Ci źle doradzić w poszukiwaniach najszybszej trasy. Mimo to nadal możemy uznać, że algorytm wytyczania trasy działa poprawnie — nawet jeśli nie można tego powiedzieć o jego danych wejściowych; dla zadanego wejścia algorytm trasujący określa trasę najszybszą.

Z kolei dla pewnych problemów ustalenie, czy dany algorytm wytwarza poprawne rozwiązanie, może się okazać trudne lub nawet niemożliwe. Jako przykład weźmy optyczne rozpoznawanie znaków. Czy ten obrazek o wymiarach  $11 \times 6$  pikseli jest cyfrą 5, czy literą S?



Niektórzy powiedzą, że to piątka, podczas gdy inni — że S, jak zatem moglibyśmy uznać poprawność lub niepoprawność decyzji komputerowej? Nie da się. W tej książce skoncentrujemy się na algorytmach komputerowych, których rozwiązania są znane.

Czasami jednak godzimy się z tym, że algorytm komputerowy produkuje niepoprawną odpowiedź, o ile tylko potrafimy panować nad tym, jak często mu się to zdarza. Dobry przykład stanowi szyfrowanie. Powszechnie używany kryptosystem RSA polega na określaniu, czy duże liczby — naprawdę duże, mające setki cyfr — są pierwsze. Jeśli zdarzyło Ci się pisać programy komputerowe, to jednym z nich był pewnie taki, który sprawdzał, czy liczba  $n$  jest pierwsza. Mógł on badać wszystkie potencjalne dzielniki od 2 do  $n - 1$  i jeśli któryś z nich okazał się rzeczywiście dzielnikiem  $n$ , to  $n$  była liczbą złożoną. Jeśli żadna liczba między 2 a  $n - 1$  nie jest dzielnikiem  $n$ , to  $n$  jest pierwsza. Jeśli jednak  $n$  ma setki cyfr, to potencjalnych dzielników jest mnóstwo — tak dużo, że nawet naprawdę szybki



komputer nie dałby rady tego sprawdzić w rozsądnym czasie. Oczywiście mógłbyś dokonać pewnych ulepszeń, na przykład wyeliminować wszystkie kandydatki parzyste po wykazaniu, że 2 nie jest dzielnikiem, lub poprzestać na dotarciu do  $\sqrt{n}$  (bo jeśli  $d$  jest większe niż  $\sqrt{n}$  i  $d$  jest dzielnikiem  $n$ , to  $n/d$  jest mniejsze niż  $\sqrt{n}$  i również jest dzielnikiem  $n$ ; jeśli więc  $n$  ma dzielnik, to znajdziesz go nim dojdiesz do  $\sqrt{n}$ ). Jeśli  $n$  ma setki cyfr, to chociaż  $\sqrt{n}$  ma około o połowę cyfr mniej niż  $n$ , nadal jest naprawdę wielką liczbą. I tu dobra wiadomość: znamy algorytm, który szybko sprawdza, czy liczba jest pierwsza. Złą wiadomością jest to, że może on popełniać błędy. W szczególności, jeśli oświadcza, że  $n$  jest złożona, to  $n$  jest na pewno złożona, lecz jeśli oświadcza, że  $n$  jest pierwsza, to istnieje pewna szansa, że  $n$  jest jednak złożona. Lecz owe złe wiadomości nie są aż tak złe: możemy utrzymywać wskaźnik błędów na naprawdę niskim poziomie, wynoszącym na przykład jeden błąd na każde  $2^{50}$  razy. Jest to na tyle rzadko — jeden błąd raz na każde sto oktylionów — że większość z nas ze spokojem przyjmuje w RSA określanie tą metodą, czy liczba jest pierwsza.

Poprawność jest delikatnym zagadnieniem w innej klasie algorytmów, nazywanych aproksymacyjnymi. Algorytmy aproksymacyjne (przybliżania pewnej wartości — przyp. tłum.) są stosowane w problemach optymalizacyjnych, w których chcemy znaleźć najlepsze rozwiązanie względem pewnej miary ilościowej. Znajdowanie najszybszej trasy, wykonywane przez GPS, jest jednym z przykładów — tu miarą ilościową jest czas podróży. Dla niektórych problemów nie mamy dobrych algorytmów znajdujących optymalne rozwiązanie w rozsądnym czasie, znamy natomiast algorytm aproksymacyjny, który w zadawalającym czasie potrafi znaleźć rozwiązanie prawie optymalne. Przez „prawie optymalne” zazwyczaj rozumiemy, że ilościowa miara rozwiązania znajdowana przez algorytm aproksymacyjny różni się pewnym znanym czynnikiem od optymalnej miary ilościowej rozwiązania. Jeśli tylko określimy, ile ów pożądaný czynnik wynosi, możemy mówić, że poprawnym rozwiązaniem algorytmu aproksymacyjnego jest każde rozwiązanie różniące się tym czynnikiem od rozwiązania optymalnego.

## Użytkowanie zasobów

Co to znaczy w odniesieniu do algorytmu *efektywne użytkowanie zasobów obliczeniowych*? O jednej z miar efektywności wspomnieliśmy, omawiając algorytmy aproksymacyjne — był nią czas. Algorytm, który daje poprawne rozwiązanie, lecz zużywa dużo czasu na jego wytworzenie, może mieć znikomą wartość lub być bezwartościowy. Gdyby Twój GPS przez godzinę wyznaczał zalecaną trasę, chciałoby Ci się go w ogóle włączać? Rzeczywiście, czas jest podstawową miarą efektywności, której używamy do oceny algorytmu, kiedy już wykazemy, że daje poprawne rozwiązanie. Lecz nie jest to miara jedyna. Może nas interesować ilość pamięci komputerowej wymaganej przez algorytm („ślady” jaki odciska w pamięci), ponieważ algorytm musi działać w dostępnej pamięci. Inne zasoby, które mogą być używane przez algorytm, to: komunikacja sieciowa, losowe bity (ponieważ algorytmy, które dokonują

losowych wyborów, potrzebują źródła liczb losowych) lub operacje dyskowe (dla algorytmów przeznaczonych do pracy z danymi przechowywanymi na dysku).

W tej książce, tak jak w większości literatury o algorytmach, koncentrujemy się tylko na jednym zasobie — jest nim czas. Jak rozsądzamy o czasie wymaganym przez algorytm? W odróżnieniu od poprawności, niezależącej od konkretnego komputera, na którym działa algorytm, faktyczny czas algorytmów zależy od kilku czynników zewnętrznych względem samego algorytmu: szybkości komputera, języka programowania, w którym algorytm jest zrealizowany, kompilatora lub interpretera tłumaczącego program na kod wykonywany w komputerze, umiejętności osoby piszącej dany program i innych działań i zdarzeń zachodzących w komputerze równoległe z wykonywanym programem. Przy tym wszystkim zakłada się, że algorytm jest wykonywany tylko przez jeden komputer, mieszczący w pamięci wszystkie jego dane.

Gdybyśmy mieli oceniać szybkość algorytmu zrealizowanego w prawdziwym języku programowania, wykonując go na konkretnym komputerze dla określonych danych i mierząc jego czas działania, nie dowiedzielibyśmy się niczego o tym, jak szybko działałby ten algorytm z danymi innego rozmiaru, a może nawet z innymi danymi o tym samym rozmiarze. A gdybyśmy chcieli porównać względną szybkość danego algorytmu z innym, dla tego samego problemu, musielibyśmy zaimplementować oba i wykonać każdy z nich z różnymi danymi, o różnych rozmiarach. Jak zatem możemy ocenić szybkość algorytmu?

Otóż robimy to, łącząc dwa pomysły. Po pierwsze, określamy, jak długo działa algorytm w funkcji rozmiaru jego danych. W przykładzie ze znajdowaniem trasy dane wejściowe byłyby pewną reprezentacją mapy drogowej, a ich rozmiar zależałby od liczby skrzyżowań i liczby dróg łączących skrzyżowania na mapie. (Fizyczny rozmiar sieci dróg nie ma znaczenia, ponieważ wszystkie odległości możemy przedstawić za pomocą liczb, a wszystkie liczby zajmują tyle samo miejsca na wejściu; długość drogi nie wpływa na rozmiar danych wejściowych). W prostszym przykładzie — przeszukiwaniu zadanej listy elementów w celu sprawdzenia, czy określony element jest na niej obecny — rozmiarem danych byłaby liczba pozycji na liście.

Po drugie, skupiamy się na tym, jak szybko funkcja charakteryzująca czas działania rośnie z rozmiarem danych wejściowych, tzn. na *tempie wzrostu* czasu działania. W rozdziale 2 zapoznamy się z notacją, której używamy do scharakteryzowania czasu działania algorytmu, lecz to, co jest w naszym podejściu najbardziej interesujące, to uwzględnianie w czasie działania tylko dominującego składnika; nie zwracamy przy tym uwagi na współczynniki. Koncentrujemy się zatem na *rzędzie wzrostu* czasu działania. Załóżmy na przykład, że udało się nam ustalić, iż dana realizacja pewnego algorytmu przeszukiwania listy  $n$  elementów zajmuje  $50n + 125$  cykli maszynowych. Składnik  $50n$  zdominuje składnik 125, jeśli  $n$  stanie się dostatecznie duże, począwszy od  $n \geq 3$  i zwiększając jeszcze tę przewagę dla list o większych rozmiarach. Wobec tego, opisując czas działania tego hipotetycznego algorytmu, nie bierzemy pod uwagę składnika 125 małego rzędu. Może się zdziwisz, ale odrzucamy też współczynnik 50. Prowadzi to do określenia czasu działania jako rosnącego liniowo

z rozmiarem danych  $n$ . Oto inny przykład: gdyby algorytm zużywał  $20n^3 + 100n^2 + 300n + 200$  cykli maszynowych, to powiedzielibyśmy, że jego czas działania rośnie jak  $n^3$ . Również w tym przypadku składniki niższego rzędu:  $100n^2$ ,  $300n$  i  $200$ , stają się coraz mniej istotne ze wzrostem rozmiaru  $n$  danych wejściowych.

W praktyce lekceważone przez nas współczynniki mają jednak znaczenie. Zależą one jednak w tak dużym stopniu od czynników zewnętrznych, że jest prawie pewne, iż przy porównywaniu dwóch algorytmów A i B, mających ten sam rząd wzrostu i działających na tych samych danych, A może działać szybciej niż B dla pewnej kombinacji maszyny, języka programowania, kompilatora (lub interpretera) i programisty, a B zadziała szybciej niż A dla innej kombinacji. Oczywiście, jeżeli oba algorytmy A i B produkują poprawne rozwiązania i A zawsze działa dwa razy szybciej niż B, to jeśli wszystko inne jest takie samo, będziemy zawsze preferowali wykonywanie A zamiast B. Jednakże z punktu widzenia abstrakcyjnego porównywania algorytmów zawsze koncentrujemy się na rzędzie wzrostu, nie przystrajając go współczynnikami lub składnikami niskiego rzędu.

I ostatnie pytanie, które stawiamy w tym rozdziale: „Dlaczego miałyby mi się opłacać zajmowanie algorytmami komputerowymi?”. Odpowiedź na nie zależy od tego, kim jesteś.

## Algorytmy komputerowe dla niekomputerowców

Nawet jeśli w sferze komputerów nie uważasz się za osobę dobrze poinformowaną, algorytmy komputerowe stanowią dla Ciebie dużą wartość. W końcu, wyjąwszy sytuację, w której podróżujesz po bezdrożach bez GPS-u, zapewne używasz go co dnia. Szukałeś czegoś dzisiaj w Internecie? W używanej przez Ciebie wyszukiwarce — Google, Bing lub jakiegokolwiek innej — są stosowane wyrafinowane algorytmy przeszukiwania Sieci i rozstrzygania, w jakiej kolejności należy przedstawiać ich wyniki. Prowadziłeś dziś samochód? O ile nie jeździsz klasycznym wehikułem, jego komputery pokładowe podjęły podczas Twojej podróży miliony decyzji, a wszystkie oparte były na algorytmach. Mógłbym tak wymieniać bez końca.

Jako docelowy użytkownik algorytmów sobie zawdzięczasz chęć dowiedzenia się czegoś o tym, jak projektujemy, charakteryzujemy i oceniamy algorytmy. Zakładam, że przejawiasz przynajmniej umiarkowane zainteresowanie, skoro ta książka trafiła do Twoich rąk i czytasz ją do tego miejsca. Powodzenia! Zobaczmy, czy uda się nam rozkręcić Cię na tyle, że na następnym spotkaniu towarzyskim dotrzymasz tonu innym, gdy rozmowa zejdzie na algorytmy<sup>2</sup>.

<sup>2</sup> No tak, przynaję, jeśli nie mieszkasz w Dolinie Krzemowej, temat algorytmów rzadko wypłynie podczas koktajli, w których bierzesz udział, niemniej z pewnych względów my, profesorowie informatyki, uważamy, że jest ważne, aby nasi studenci nie konsternowali nas na spotkaniach towarzyskich brakiem wiedzy z poszczególnych dziedzin informatyki.

## Algorytmy komputerowe dla komputerowców

Jeśli jesteś za pan brat z komputerami, to masz większe obowiązki wobec algorytmów! One nie tylko znajdują się w centrum wszystkiego, co się dzieje w Twoim komputerze — algorytmy są technologią, i to równie istotną jak wszystko, co współtworzy Twój komputer. Możesz zapłacić dodatkową cenę za komputer z najnowszym i największym procesorem, lecz aby te pieniądze okazały się trafionym wydatkiem, potrzebujesz, by w tym komputerze działały realizacje dobrych algorytmów.

Oto przykład ilustrujący, że algorytmy rzeczywiście stanowią technologię. W trzecim rozdziale przyjrzymy się trzem różnym algorytmom sortowania w porządku rosnącym listy  $n$  wartości. Niektóre z tych algorytmów będą osiągały czasy działania rosnące jak  $n^2$ , lecz inne będą działać w czasie rosnącym tylko jak  $n \lg n$ . Co to jest  $\lg n$ ? Jest to logarytm przy podstawie 2 z  $n$ , czyli  $\log_2 n$ . Informatycy używają logarytmów z podstawą 2 tak często, że podobnie jak matematycy i naukowcy, którzy dla skrócenia stosują zapis  $\ln n$  na oznaczenie logarytmu naturalnego —  $\log_e n$ , również oni używają własnego skrótu dla logarytmów przy podstawie 2. I tak, ponieważ funkcja  $\lg n$  jest odwrotnością funkcji wykładniczej, rośnie ona bardzo powoli z  $n$ . Jeśli  $n = 2^x$ , to  $x = \lg n$ . Na przykład  $2^{10} = 1024$ , więc  $\lg 1024$  wynosi tylko 10, podobnie  $2^{20} = 1\,048\,576$ , zatem  $\lg 1\,048\,576$  wynosi zaledwie 20, a  $2^{30} = 1\,073\,741\,824$ , co oznacza, że  $\lg 1\,073\,741\,824$  wynosi raptem 30. Tak więc we wzroście  $n \lg n$  w porównaniu ze wzrostem  $n^2$  zachodzi wymiana czynnika  $n$  na jedynie  $\ln n$ , a to już jest gra warta świeczki.

Skonkretyzujmy nieco bardziej ten przykład, wystawiając szybszy komputer (komputer A) wykonujący algorytm sortowania, którego czas działania dla  $n$  wartości rośnie jak  $n^2$ , przeciw wolniejszemu komputerowi B, wykonującemu algorytm sortowania w czasie rosnącym jak  $n \lg n$ . Każdy z komputerów musi posortować 10 milionów liczb. (Choć może się wydawać, że 10 milionów liczb to dużo, jeśli są to 8-bajtowe liczby całkowite, ich rozmiar na wejściu zajmie około 80 megabajtów, co zmieści się w pamięci nawet niedrogiego laptopa sprzed wielu lat). Przypuśćmy, że komputer A wykonuje 10 miliardów operacji na sekundę (jest szybszy od każdego sekwencyjnego komputera w chwili, gdy piszę te słowa), a komputer B wykonuje tylko 10 milionów operacji na sekundę, czyli komputer A jest 1000 razy szybszy od komputera B, biorąc pod uwagę samą moc obliczeniową. Aby powiększyć jeszcze różnicę, założmy, że światowej sławy programistka koduje komputer A w języku maszynowym, a kod wynikowy wymaga do posortowania  $n$  liczb  $2n^2$  rozkazów. Założmy dalej, że program dla komputera B pisze programista zupełnie przeciętny, korzystając z języka wysokiego poziomu i mało efektywnego kompilatora, w wyniku czego powstaje kod złożony z  $50n \lg n$  rozkazów. Aby posortować 10 milionów liczb, komputer A zużywa

$$\frac{2 \cdot (10^7)^2 \text{ rozkazów}}{10^{10} \text{ rozkazów / sekundę}} = 20\,000 \text{ sekund,}$$

co stanowi więcej niż 5 i pół godziny, podczas gdy komputerowi B zabiera to

$$\frac{50 \cdot 10^7 \lg 10^7 \text{ rozkazów}}{10^7 \text{ rozkazów / sekundę}} \approx 1163 \text{ sekundy}$$

— równowartość mniej niż 20 minut. Stosując algorytm, którego czas działania rośnie znacznie wolniej, nawet z marnym kompilatorem, komputer B działa 17 razy szybciej niż komputer A! Przewaga algorytmu  $n \lg n$  byłaby jeszcze wyraźniejsza, gdybyśmy sortowali 100 milionów liczb. Komputerowi A zabrałoby to ponad 23 dni, algorytm  $n \lg n$  na komputerze B uporałby się z robotą w cztery godziny. Uogólniając, wraz ze wzrostem rozmiaru problemu rośnie względna przewaga algorytmu  $n \lg n$ .

Nawet mimo imponującego postępu, który nieustannie obserwujemy w sprzęcie komputerowym, całościowa produktywność systemu zależy w równym stopniu od doboru efektywnych algorytmów, co od doboru szybkiego sprzętu lub sprawnie działających systemów operacyjnych. Błyskawiczny postęp, jaki dokonał się w innych technologiach komputerowych, dotyczy także algorytmów.

## Co czytać dalej

W moim bardzo subiektywnym odczuciu najklarowniejszym i najbardziej użytecznym źródłem wiedzy o algorytmach komputerowych jest *Wprowadzenie do algorytmów* [CLRS09], napisane przez czterech diabelnie przystojnych facetów. Książka ta jest powszechnie określana jako „CLRS”, od inicjałów autorów. Sięgałem do niej po większość materiału pomieszczonego w tej książce. Jest ona nieporównanie bardziej kompletna od tej książki, lecz zakłada się w niej, że masz choć trochę doświadczenia w programowaniu komputerów i jesteś za pan brat z matematyką. Jeśli uznasz, że odpowiada Ci poziom matematyczny niniejszej książki, i masz odwagę ruszyć głębiej w temat, nie możesz postąpić lepiej, niż sięgnąć po CLRS. (W mojej skromnej opinii, ma się rozumieć).

Książka Johna MacCormica *Nine Algorithms That Changed the Future* [Mac12] zawiera opis kilku algorytmów i związanych z nimi kwestii obliczeniowych, które oddziałują na nasze codzienne życie. Ujęcie zastosowane przez MacCormica jest mniej techniczne niż w tej książce. Jeśli dojdiesz do wniosku, że tutaj było „zbyt matematycznie”, to polecam Ci skosztowanie lektury MacCormica. Zdołasz przyswoić z niej wiele, nawet jeśli masz wątle przygotowanie matematyczne.

W mało prawdopodobnym przypadku, gdy uznasz, że CLRS jest zbyt rozwodnione, możesz spróbować zaczerpnąć z wielotomowego zbioru Donalda Knutha *The Art of Computer Programming* [Knu97, Knu98a, Knu98b, Knu11]. Chociaż tytuł serii sugeruje, że skupia się ona na pisaniu kodu, książki te zawierają wspaniałą, głęboką analizę algorytmów<sup>3</sup>. Ostrzegam jednak: materiał zawarty w *TAOCP* sięga głęboko. Przy okazji, jeśli zastanawiasz się, skąd się wzięło słowo „algorytm”, to Knuth powiada, że pochodzi ono od imienia „al-Khowârizmî” perskiego matematyka z IX wieku.

<sup>3</sup> *Sztuka programowania*, WNT, Warszawa 2002 – 2007. Jak dotąd ukazały się 4 książki z jeszcze nie ukończonego cyklu, który Donald Knuth zaplanował na 7 tomów — *przyp. tłum.*

Oprócz CLRS przez lata ukazało się kilka innych świetnych tekstów o algorytmach komputerowych. Noty do rozdziału 1 w CLRS zawierają odwołania do wielu takich tekstów. Zamiast powielać je tutaj, kieruję Cię wprost do CLRS.

---

# Skorowidz

## A

abstract data type, *Patrz:* ADT  
abstrahowanie, 107  
adjacency matrix, *Patrz:* macierz sąsiedztwa  
adjacency-list representation, *Patrz:* lista sąsiedztwa reprezentacja  
Adleman Leonard, 153  
ADT, 107  
Advanced Encryption Standard, *Patrz:* AES  
AES, 150, 160  
algorytm, 15, 21  
  aproxymacyjny, 17, 208  
  Bellmana-Forda, 102, 111, 183, 186  
    czas działania, *Patrz:* czas działania algorytmu Bellmana-Forda  
    czas działania, *Patrz:* czas działania  
  Dijkstry, 102, 104, 105, 107, 110, 169, 210  
    czas działania, 106  
  Euklidesa, 157  
  Floyda-Warshalla, 102, 115, 116, 117, 118, 122, 210  
    czas działania, 116, 121  
  KMP, 143  
  Knutha-Morrisa-Pratta, 143  
  komputerowy, 19, 20  
    opisywanie, 23  
  o czasie wielomianowym, *Patrz:* algorytm wielomianowy przybliżający, *Patrz:* algorytm aproksymacyjny  
  redukcji w czasie wielomianowym, 187, 189  
  rekurencyjny, 35  
  sortowania, *Patrz:* sortowanie  
  szybkość, 18  
  wielomianowy, 183, 184, 186, 187, 193, 194, 196, 202, 210  
  zachłanny, 169  
  znajdowania najkrótszej ścieżki, 94  
all-pairs shortest-paths, *Patrz:* problem najkrótszych ścieżek między wszystkimi parami wierzchołków

alternatywa  
  wykluczająca, *Patrz:* XOR  
approximation algorithm, *Patrz:* algorytm aproksymacyjny  
arbitrage opportunity, *Patrz:* możliwość arbitrażu  
arbitraż, *Patrz:* możliwość arbitrażu  
array, *Patrz:* tablica  
arytmetyka  
  modularna, 153, 156  
  modulo, *Patrz:* arytmetyka modularna  
assigns, *Patrz:* procedura przypisywanie  
authentication, *Patrz:* uwierzytelnienie  
automat skończony, 138, 139, 141

## B

Bacon Kevin, 101  
base cases, *Patrz:* przypadek bazowy  
Bellman Richard, 111  
Bellmana-Forda algorytm, *Patrz:* algorytm Bellmana-Forda  
binary heap, *Patrz:* kopiec binarny  
binary tree, *Patrz:* drzewo binarne  
biologia obliczeniowa, 125, 131  
block cipher, *Patrz:* szyfr blokowy  
bloczek jednorazowy, *Patrz:* podkładka jednorazowa  
Boole George, 190  
boolean formula, *Patrz:* formuła boolowska  
boolean formula satisfiability problem, *Patrz:* problem spełnialności formuły boolowskiej  
branch and bound method, *Patrz:* metoda podziału i ograniczeń

## C

certificate, *Patrz:* certyfikat  
certyfikat, 185, 186, 195, 197  
CGF, 210  
child, *Patrz:* dziecko  
ciąg, 125  
cipher block chaining, *Patrz:* łańcuchowanie bloków szyfru

ciphertext, *Patrz:* tekst zaszyfrowany  
 clause, *Patrz:* klauzula  
 common subsequence, *Patrz:* podciąg  
     wspólny  
 comparison sort, *Patrz:* sortowanie  
     porównanie  
 complete graph, *Patrz:* graf pełny  
 composite number, *Patrz:* liczba złożona  
 concatenation, *Patrz:* złączenie  
 connected graph, *Patrz:* graf spójny  
 contex-free grammar problem, *Patrz:* CGF  
 counting sort, *Patrz:* sortowanie zliczanie  
 critical path, *Patrz:* ścieżka krytyczna  
 cykl, 87, 94, 101  
     Eulera, 183, 184, 186  
     Hamiltona, 183, 184, 185, 186, 196,  
         197, 198, 204, 205, 206  
     z wagą ujemną, 113  
 czas, 17, 18  
 działania, 20  
     algorytmu Bellmana-Forda, 113, 183  
     algorytmu Dijkstry:, 106  
     algorytmu Floyda-Warshalla, 116, 121  
     algorytmu redukcji, 187  
     dopasowywania napisów, 143  
     notacja, 23  
     ograniczenie, *Patrz:* ograniczenie  
     ograniczony przez funkcję liniową, 30  
     operacji, 29  
     rosnący liniowo, 18  
     rząd wzrostu, 18, 19  
     tempo wzrostu, 18  
     wielomianowy, *Patrz:* algorytm  
         wielomianowy  
     wyszukiwanie binarne, *Patrz:*  
         wyszukiwanie binarne czas działania  
 liniowy, 210  
 podliniowy, 210  
 sortowania, 38  
     pozycyjne, 71  
     scalanie, 50, 58, 59, 67, 71  
     szybkie, 59, 65, 67, 71  
     topologiczne, 92  
     wstawianie, 49, 50, 67, 71  
     wybieranie, 45, 46, 50, 67, 71  
     zliczanie, 71, 78  
 weryfikacji certyfikatu, 185  
 wielomianowy, *Patrz:* algorytm  
     wielomianowy

## D

dag, 87, 92, 96, 101  
     sortowanie topologiczne, *Patrz:*  
         sortowanie topologiczne  
 dane  
     dekompresja, *Patrz:* dekompresja  
     kompresja, *Patrz:* kompresja  
     nadmiarowe, 164  
     satelickie, *Patrz:* dane towarzyszące  
     struktura, 107, 179  
     tablica, *Patrz:* tablica  
     towarzyszące, 38, 39, 71, 73, 78  
     typ abstrakcyjny, *Patrz:* ADT  
     wejściowe, 24  
     kopia, 51  
     rozmiar, 18, 29  
     wyjściowe, *Patrz:* wynik  
 data structure, *Patrz:* dane struktura  
 decidable problem, *Patrz:* problem podatny  
 decision problems, *Patrz:* problem  
     decyzyjny  
 decryption, *Patrz:* deszyfrowanie  
 dekompresja, 163, 165, 169, 170  
     LZW, 176, 177  
 dense graph, *Patrz:* graf gęsty  
 deszyfrowanie, 145  
     długiego komunikatu, 160  
 diagram, 83, 87  
     PERT, 92, 93, 100  
         z zanegowanymi czasami, 95  
 digraf, *Patrz:* graf skierowany  
 Dijkstra Edsger, 102  
 directed acyclic graph, *Patrz:* dag  
 directed edge, *Patrz:* krawędź skierowana  
 directed graph, *Patrz:* graf skierowany  
 divide-and-conquer, *Patrz:* metoda dziel  
     i zwyciężaj  
 DNA, 125, 164  
 doubly linked list, *Patrz:* lista powiązana  
     dwukierunkowa  
 droga Eulera, *Patrz:* cykl Eulera  
 drzewo binarne, 108, 109, 166  
     aktualizacja, 170  
 dynamic programming, *Patrz:*  
     programowanie dynamiczne  
 dziecko, 109



**E**

efektywność, *Patrz:* rozwiązanie efektywne  
miara, 17  
element  
osiowy, 59, 62, 63  
rozdzielający, *Patrz:* element osiowy  
uporządkowanie, *Patrz:* uporządkowanie  
encryption, *Patrz:* szyfrowanie  
entry, *Patrz:* wpis  
Erdős Paul, 101  
escape code, *Patrz:* kod sygnałowy  
Euler Leonhard, 183  
Euler tour, *Patrz:* cykl Eulera  
Eulera cykl, *Patrz:* cykl Eulera  
exclusive-or, *Patrz:* XOR

**F**

faks, 164, 170  
Fermata twierdzenie, *Patrz:* twierdzenie  
Fermata małe  
finite automaton, *Patrz:* automat  
skończony  
F-kopiec, *Patrz:* kopiec Fibonacciego  
Floyd Robert, 116  
Ford Lester, 111  
formal language, *Patrz:* język formalny  
format  
JPEG, *Patrz:* MP3  
MP3, *Patrz:* MP3  
formuła  
boolowska, 190, 199, 205  
postać 3-koniunkcyjną normalna, 191  
koniunkcja klauzul, 191  
logiczna, *Patrz:* formuła boolowska  
spełnialna, 190  
spełnialności, 189  
funkcja, *Patrz:* procedura

**G**

gadżet, 197, 206  
generator liczb pseudolosowych, *Patrz:*  
PRNG  
graf, 186  
acykliczny, 207  
cykliczny, 101  
gęsty, 110, 116

nieskierowany, 183, 192, 195, 199  
pełny, 198  
rzadki, 110, 116  
skierowany, 87  
acykliczny, *Patrz:* dag  
reprezentacja, 90  
ważony, 94, 95, 101  
spójny, 183  
nieskierowany, 183, 184  
grafika komputerowa, 38  
grafu, 108  
gramatyka bezkontekstowa, 210  
greedy algorithm, *Patrz:* algorytm  
zachłanny

**H**

halting problem, *Patrz:* problem stopu  
Hamilton William, 184  
hamiltonian cycle, *Patrz:* cykl Hamiltona  
hamiltonian-path problem, *Patrz:* problem  
ścieżki Hamiltona  
hash table, *Patrz:* tablica z haszowaniem  
heap property, *Patrz:* klucz własność kopca  
heapsort, *Patrz:* sortowanie na kopcu  
Huffman David, 165  
Huffmana kod, *Patrz:* kod Huffmana

**I**

in place, *Patrz:* sortowanie na miejscu  
incident edge, *Patrz:* krawędź incydentna  
in-degree, *Patrz:* stopień wejściowy  
initialization vector, *Patrz:* wektor  
początkowy  
insertion sort, *Patrz:* sortowanie  
wstawianie  
internal node, *Patrz:* węzeł wewnętrzny  
iteracja, 26, 43, 49

**J**

język  
formalny, 210  
programowania, 156  
Python, *Patrz:* Python  
Juliusz Cezar, 146

**K**

karta kredytowa, 145, 147  
 key, *Patrz:* klucz  
 klauzula, 191, 194, 199, 201  
 klika, 192, 194  
   problem, 192  
   rozmiar, 192  
 klucz, 37, 71, 74  
   jawny, *Patrz:* klucz publiczny  
   kryptograficzny, 146  
   podkładka, 149  
   potomka, 109  
   prywatny, *Patrz:* klucz tajny  
   publiczny, 151, 160  
   symetryczny, 147, 150, 160  
   tajny, 151, 152  
   węzła, 109  
   własność kopca, 109  
 knapsack problem, *Patrz:* problem plecakowy  
 Knuth Donald, 21  
 kod  
   bezprzedrostkowy, 165, 166  
   Huffmana, 165, 171, 179  
   adaptacyjny, 170  
   seria-długość, 170  
   sygnałowy, 170  
 kolejka priorytetowa, 107  
 kompresja, 163, 170  
   bezstratna, 163, 164, 171  
   LZW, 171, 173, 176, 179  
   stratna, 163, 164  
 koniunkcja, 191  
 konkatenacja, *Patrz:* złączenie  
 kopiec  
   binarny, 109  
   Fibonacciego, 111  
 krawędź, 91, 183  
   incydentna, 183, 184  
   o ujemnych wagach, 102  
   osłabianie, *Patrz:* osłabianie skierowana, 87  
   waga, 94  
 krok osłabiający, 97  
 kryptografia, 145  
   z kluczem jawnym, 151, 153  
   z kluczem symetrycznym, 147, 150

kryptosystem  
   hybrydowy, 151, 160  
   RSA, 145, *Patrz:* RSA  
   z kluczem jawnym, 153, 159  
 kubekowanie, *Patrz:* szufladkowanie

**L**

last in first out, *Patrz:* porządek LIFO  
 LCS, 125, 126, 127, 129  
 leaf, *Patrz:* liść  
 lexicographic ordering, *Patrz:*  
   uporządkowanie leksykograficzne  
 liczba  
   Bacona, 101  
   Erdösa, 102  
   Erdösa-Bacona, 102  
   losowa, 161  
   pierwsza, 16, 153, 154  
   duża, 156  
   twierdzenie, 156  
   względnie pierwsza, 155, 157  
   złożona, 154  
 linear search, *Patrz:* wyszukiwanie liniowe  
 linked list, *Patrz:* lista powiązana  
 lista, 23  
   powiązana, 91, 92  
   dwukierunkowa, 92  
   jednokierunkowa, 92  
   sąsiedztwa, 91  
   reprezentacja, 91, 92  
   z dowiązaniem, *Patrz:* lista powiązana  
 liść, 109, 166  
   głębokość, 167  
 literal, *Patrz:* literał  
 literał, 191, 194, 199  
 logarytm, 20  
   naturalny, 20  
   przy podstawie 2, 20  
*longest common subsequence*, *Patrz:* LCS  
*longest-acyclic-path problem*, *Patrz:*  
   problem najdłuższej ścieżki prostej  
 loop, *Patrz:* pętla  
 loop body, *Patrz:* pętla treść  
 loop invariant, *Patrz:* pętla niezmiennik  
 loop variable, *Patrz:* zmienna pętli  
 lossless compression, *Patrz:* kompresja bezstratna  
 lossy compression, *Patrz:* kompresja stratna

**L**

łańcuch, *Patrz:* napis  
 łańcuchowanie bloków szyfru, 150  
 huk, *Patrz:* krawędź skierowana

**M**

MacCormic John, 21  
 macierz sąsiedztwa, 90, 91  
 merge sort, *Patrz:* sortowanie scalanie  
 metoda, *Patrz:* procedura  
   2-opt, 207  
   dziel i zwyciężaj, 51, 59  
   podziału i ograniczeń, 207  
   przeszukiwanie sąsiedztwa, 207  
   rekurencji uniwersalnej, 59  
 modular arithmetic, *Patrz:* arytmetyka  
 modularna  
 Mother Problem, *Patrz:* problem matka  
 możliwość arbitrażu, 115

**N**

nadmiarowość, 164  
 napis, 125, 209  
   dopasowywanie, 125, 137, 141, 143  
   czas działania, 143  
   wzorcowy, 137  
   z tekstem, 137  
   zamiana na inny, 125, 130, 131, 132  
   złączenie, *Patrz:* złączenie  
 neighborhood search, *Patrz:* metoda  
   przeszukiwanie sąsiedztwa  
 nierówność trójkąta, 208  
 node, *Patrz:* węzeł, wierzchołek  
 notacja  
    $\Theta$ , 30, 31, 32, 43, 45, 73, 116  
   asymptotyczna, 32, 46  
    $O$ , 31, 32, 43  
 NP-complete problem, *Patrz:* problem NP-  
 zupełny  
 NP-hard problem, *Patrz:* problem NP-trudny

**O**

ograniczenie  
   dolne, 32, 45, 46, 73, 78  
   egzystencjalne, 73  
   uniwersalne, 73  
 górne, 30, 31, 32, 45, 46

one-time pad, *Patrz:* podkładka jednorazowa  
 operacja  
   alternatywy wykluczającej, 148, 149  
   czas działania, 29  
   koszt, 131, 134  
 operator  
   „i”, 48  
   krótko spinający, 48  
 optymalizacja, 17  
 osłabianie, 97, 98, 104, 107, 111, 169  
 out-degree, *Patrz:* stopień wyjściowy  
 output, *Patrz:* wynik

**P**

pad, *Patrz:* podkładka  
 paradygmat  
   dziel i zwyciężaj, *Patrz:* metoda dziel  
   i zwyciężaj  
 paradygmat algorytmiczny, *Patrz:* algorytm  
 paradygmat  
 parametr, 24  
 partition problem, *Patrz:* problem podziału  
 partitioning, *Patrz:* rozdzielanie  
 path, *Patrz:* ścieżka  
 pattern string, *Patrz:* napis wzorcowy  
 PCP, 209, 210  
 peł, 164, 171  
 pętla, 26  
   niezmiennik, 33, 45, 105, 106  
   inicjowanie, 33, 42  
   utrzymanie, 33, 42  
   zakończenie, 33, 42  
 treść, 26  
 zmienna, 26  
 pivot, *Patrz:* element osiowy  
 plaintext, *Patrz:* tekst jawny  
 podciąg, 125  
   wspólny, 126  
   najdłuższy, *Patrz:* LCS  
 podkładka, 149  
   jednorazowa, 147, 149  
 podnapis, 126  
 podnoszenie do kwadratu powtarzane, 159  
 podstruktura optymalna, 122, 126, 127  
 podścieżka, 117  
 podtablica, 40, 47  
 pokrycie wierzchołkowe, 195, 196, 206  
   problem, 195  
   rozmiar, 195

- polynomial-time algorithm, *Patrz:* algorytm wielomianowy
- polynomial-time reduction algorithm, *Patrz:* algorytm redukcji w czasie wielomianowym
- poprawność, *Patrz:* rozwiązanie poprawne
- poprzednik, 97, 118
- porządek, 84
  - LIFO, 89
  - liniowy, 87
  - ostatni przychodzi, pierwszy wychodzi, *Patrz:* porządek LIFO
- Post's correspondence problem, *Patrz:* PCP
- postęp arytmetyczny, 45
- potomek, 108
  - klucz, 109
- predecessor, *Patrz:* poprzednik
- prefix, *Patrz:* przedrostek
- prefix-free code, *Patrz:* kod bezprzedrostkowy
- prime number, *Patrz:* liczba pierwsza
- priority queue, *Patrz:* kolejka priorytetowa
- PRNG, 161
- problem
  - cyklu Hamiltona, *Patrz:* cykl Hamiltona
  - decyzyjny, 186, 187
  - dopasowywania napisów, 125, 137, 141, 143
  - gramatyki bezkontekstowej, *Patrz:* CGF
  - kliki, 192, 195, 206
  - komiwojażera, 182, 197, 198, 204, 205, 206, 207, 208
  - matka, 189, 191, 205
  - najdłuższej ścieżki acyklicznej, *Patrz:*
    - problem najdłuższej ścieżki prostej
    - najdłuższej ścieżki prostej, 199
    - najkrótszych ścieżek między wszystkimi parami wierzchołków, 115
  - nierozstrzygalny, 208, 209, 210
  - NP-trudny, 186, 187, 189, 197, 198, 203, 204
  - NP-zupełny, 183, 184, 185, 186, 187, 189, 192, 197, 204, 205, 206, 207, 208
  - odpowiedniości Posta, *Patrz:* PCP
  - optymalizacyjny, 186, 204, 207
  - plecakowy, 204
  - podatny, 184, 185
  - podziału, 203, 204
  - pokrycia wierzchołkowego, 195, 206
  - spełnialności 3-CNF, 192, 193
  - spełnialności formuły boolowskiej, 190, 191
  - stopu, 208
  - sumy podzbioru, 199, 203
  - ścieżki Hamiltona, 197
  - w klasie NP, 185, 186, 187, 198
  - w klasie P, 184, 185
  - zatrzymania, *Patrz:* problem stopu
- procedura, 24
  - automat skończony, 141
  - Bellmana-Forda, 111
  - cykl z wagą ujemną, 114
  - dekompresja LZW, 177
  - Dijkstra, 104
  - dopasowywania napisów, 141
  - drzewo Huffmana, 168, 169
  - Euklidesa, 157
  - Floyda-Warshalla, 119
  - kompresja LZW, 174, 179
  - LCS, 128
    - rekurencyjna, 129
  - parametr, *Patrz:* parametr
  - przypisywanie, 26
  - rozdzielanie, 63, 66
  - sortowanie
    - na kopcu, 110
    - scalanie, 51, 52, 57
    - szybkie, 60
    - topologiczne, 88
    - wstawianie, 47
    - wybijanie, 44
  - wyszukiwanie binarne, 41
  - wywołanie, 24
- program deterministyczny, 161
- programowanie dynamiczne, 121, 122, 123, 126
- przedrostek, 126, 142, 165
- przeglądarka, 145
- przegródka, 39
- przepływ sterowania, 23
- przeszukiwanie sąsiedztwa, *Patrz:* metoda
  - przeszukiwanie sąsiedztwa
- przypadek bazowy, 35, 51
- przyrostek, 142
- pseudokod, 23
- pseudorandom number generator, *Patrz:* PRNG
- public key, *Patrz:* klucz publiczny

public key cryptography, *Patrz:* kryptografia z kluczem jawnym

## Q

quicksort, *Patrz:* sortowanie szybkie

## R

recurrence equation, *Patrz:* rekurencja

recursion, *Patrz:* rekursja

reduction, *Patrz:* redukcja

redukcja, 187, 188, 191, 194, 197, 202, 203, 204, 205

rekurencja, 58

uniwersalna, 59, 65

rekursja, 34, 52

relacja przechodnia, 83

relaksacja, *Patrz:* osłabianie

relative prime number, *Patrz:* liczba względnie pierwsza

relaxation step, *Patrz:* krok osłabiający

Rivest Ronald, 153

rozdzielanie, 60, 62

procedura, 63, 66

rozwiązanie

efektywne, 16, 17

oszczędne, *Patrz:* rozwiązanie efektywne

poprawne, 16, 17

prawie optymalne, 17

RSA, 16, 153, 154, 156, 159

run-length encoding, *Patrz:* kod seriadługość

## S

satellite data, *Patrz:* dane towarzyszące

satisfiable formula, *Patrz:* formuła spełnialna

secret key, *Patrz:* klucz tajny

seed, *Patrz:* ziarno

selection sort, *Patrz:* sortowanie wybieranie

sentinel, *Patrz:* wartownik

sequence, *Patrz:* ciąg

Shamir Adi, 153

shift cipher, *Patrz:* szyfr z przesunięciem

short circuiting, *Patrz:* operator krótko spinający

shortest of path, *Patrz:* ścieżka najkrótsza

sieci średnica, 116

simple substitution cipher, *Patrz:* szyfr prosty podstawieniowy

single-pair shortest path, *Patrz:* ścieżka najkrótsza między jedną parą wierzchołków

single-source shortest paths, *Patrz:* ścieżka najkrótsza z jednym źródłem

singly linked list, *Patrz:* lista powiązana jednokierunkowa

slot, *Patrz:* przegródka

sort key, *Patrz:* sortowanie klucz

sortowanie, 39

binarne, 210

czas działania, 38, 45, 46, 49, 50, 59, 62, 65, 67, 71, 78, 92, 210

deterministyczne, 68

klucz, 38, 39

na kopcu, 110

na miejscu, 51, 56, 59

porównanie, 72

pozycyjne, 79, 80, 81, 210

czas działania, 71

scalanie, 38, 50, 58, 67, 210

czas działania, 50, 58, 59, 67, 71

stabilne, 79, 80

szybkie, 38, 59, 62, 67, 210

czas działania, 59, 65, 67, 71

topologiczne, 87, 88, 89, 210

czas działania, 92

w porządku rosnącym, 20

wstawianie, 38, 46, 47, 50, 67, 210

czas działania, 49, 50, 67, 71

wybijanie, 38, 43, 46, 67, 210

czas działania, 45, 46, 50, 67, 71

zliczanie, 79, 80, 81, 210

czas działania, 71, 78

procedura, 78

source vertex, *Patrz:* wierzchołek źródłowy

sparse graph, *Patrz:* graf rzadki

spełnialność

3-CNF, 191, 192, 199, 205, 206

stack, *Patrz:* stos

stan, 139

state, *Patrz:* stan

Stinson Douglas, 146

stopień

wejściowy, 88, 90, 94

wyjściowy, 88, 94

stos, 89  
 string, *Patrz:* napis  
 string matching, *Patrz:* problem dopasowywania napisów  
 subsequence, *Patrz:* podciąg  
 subset-sum problem, *Patrz:* problem sumy podzbioru  
 substring, *Patrz:* podnapis  
 suffix, *Patrz:* przyrostek  
 symbol  $\oplus$ , 148  
 symmetric-key cryptography, *Patrz:* kryptografia z kluczem symetrycznym  
 system AES, *Patrz:* AES  
 szufladkowanie, 39  
 szyfr  
 blokowy, 149, 150  
 łańcuchowanie bloków, *Patrz:* łańcuchowanie bloków szyfru  
 prosty podstawieniowy, 146  
 z kluczem symetrycznym, 147, 150  
 z przesunięciem, 146  
 szyfrowanie, 16, 145  
 długiego komunikatu, 160  
 standard zaawansowany, *Patrz:* AES

## Ś

ścieżka, 93  
 krytyczna, 92, 93, 94  
 najkrótsza, 94, 95, 96  
 między jedną parą wierzchołków, 101  
 między wszystkimi parami wierzchołków, 115  
 podścieżka, 117, 118  
 waga, 117  
 z jednym źródłem, 97, 101  
 waga, 95, 117  
 średnica sieci, 116  
 świadectwo, *Patrz:* certyfikat

## T

tabela indeks pozycji, *Patrz:* wpis  
 tablica, 24, 40, 90, 107  
 błąd indeksowania, 48  
 element, 24  
 indeksowanie, 78  
 odwrotnie uporządkowana, 67  
 permutowanie, 43

posortowana, 37, 38, *Patrz też:* sortowanie  
 prawie posortowana, 50  
 przeszukiwanie, 25  
 z haszowaniem, 179  
 target vertex, *Patrz:* wierzchołek docelowy  
 tekst  
 jawny, 146, 149  
 kostkowanie bloków, 150  
 plasterkowanie bloków, 150  
 zaszyfrowany, 146, 149, 152  
 test prymarności  
 AKS, 154  
 Millera-Rabina, 154, 156  
 oparty na małym twierdzeniu Fermata, 159  
 text string, *Patrz:* napis z tekstem  
 traveling-salesman problem, *Patrz:* problem komiwojażera  
 triangle inequality, *Patrz:* nierówność trójkąta  
 trie, 179  
 Turing Alan, 208  
 twierdzenie  
 Fermata małe, 156, 160  
 o liczbach pierwszych, 156

## U

undecidable problem, *Patrz:* problem nierozstrzygalny  
 undirected graph, *Patrz:* graf nieskierowany  
 uporządkowanie, 37  
 leksykograficzne, 37  
 uwierzytelnienie, 145

## V

variable, *Patrz:* zmienna  
 vertex, *Patrz:* wierzchołek  
 vertex cover, *Patrz:* pokrycie wierzchołkowe  
 vertex cover problem, *Patrz:* pokrycie wierzchołkowe problem  
 vertex degree, *Patrz:* wierzchołek stopień

**W**

Warshall Stephen, 116  
 wartościowanie, 192  
 wartownik, 28  
 weight, *Patrz:* krawędź waga  
 weight of path, *Patrz:* ścieżka waga  
 weighted directed graph, *Patrz:* graf skierowany ważony  
 wektor  
   początkowy, 150  
 wetware, 23  
 węzeł, 108, *Patrz:* wierzchołek  
   klucz, 109  
   wewnętrzny, 166  
 wierzchołek, 87  
   docelowy, 96  
   stopień, 184  
   źródłowy, 96  
 wpis, 24

wyszukiwanie

  binarne, 37, 39, 40, 41, 67  
     czas działania, 39, 43, 67  
     zapis rekurencyjny, 42  
   liniowe, 25, 26, 27, 67, 210  
     czas działania, 29, 30, 31, 67  
     z wartownikiem, 28, 29, 32  
     zapis rekurencyjny, 35

**X**

XOR, 148, 149

**Z**

zależność rekurencyjna, *Patrz:* rekurencja  
 zasoby obliczeniowe, 16, 17  
 ziarno, 161  
 złączenie, 142  
 zmienna, 26, 27  
   pętli, 26  
 znak  
    $\oplus$ , 148  
   zdekodowany, 170





# PROGRAM PARTNERSKI

— GRUPY HELION —

1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

<http://program-partnerski.helion.pl>

GRUPA  
**Helion** 

# DOWIEDZ SIĘ, JAK DZIAŁAJĄ APLIKACJE KOMPRESUJĄCE I SZYFRUJĄCE!

Każdy program działa według określonego algorytmu — Twoja nawigacja GPS, system płatności elektronicznych, wyszukiwarka Google. Algorytmy są jak przepisy kucharskie: zrób to, sprawdź tamto. Jednak konsekwencje popełnienia błędu w algorytmie są zupełnie inne niż w przypadku niesprawdzonego przepisu. To właśnie algorytmy decydują o czasie wykonania skomplikowanych operacji przez programy komputerowe, a ich odpowiednia lub nieodpowiednia implementacja może sprawić, że Twój projekt wart miliony odniesie sukces lub poniesie porażkę.

Dzięki tej książce będziesz mógł bezboleśnie wkroczyć w świat algorytmów. W trakcie lektury dowiesz się, czym są algorytmy, jak się je projektuje i prezentuje. Po wstępie teoretycznym poznasz najpopularniejsze algorytmy sortowania i wyszukiwania, algorytmy znajdowania najkrótszej ścieżki oraz algorytmy operujące na ciągach znaków. Następnie przejdziesz do najciekawszych zagadnień związanych z kryptografią i kompresją danych. Zastanawiasz się, czy są miejsca, w których znane algorytmy nie radzą sobie zbyt dobrze? To problemy NP-zupełne — z nimi też będziesz mógł się zaznajomić. Ta książka jest interesującym przewodnikiem po świecie algorytmów, a zarazem przyjemną lekturą dla każdego programisty i pasjonata informatyki.

## Poznaj algorytmy:

- sortujące i wyszukiujące
- znajdowania najkrótszej ścieżki
- kryptograficzne
- kompresujące

	<i>Sprawdź nasze szkolenia!</i>	<b>KOD KORZYŚCI</b> <i>Sięgnij po więcej!</i>	
 <a href="http://helion.pl">helion.pl</a>		ISBN 978-83-283-6736-4	
 <b>HELION SA</b> ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	<b>AKADEMIA IT &amp; BUSINESS</b>		
<a href="http://WWW.SZKOLENIA.HELION.PL">WWW.SZKOLENIA.HELION.PL</a>		9 788328 367364	
<b>INFORMATYKA W NAJLEPSZYM WYDANIU</b>		Cena: 54,90 zł	