

Feliks Kurp

ALGORYTMY

Struktury danych
i złożoność obliczeniowa



Helion

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.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Szymon Szwajger

Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdriowski, Zaprucki

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <https://helion.pl> (księgarnia internetowa, katalog książek)

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

ISBN: 978-83-283-9530-5
Copyright © Helion S.A. 2022

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści

	Wstęp	7
ROZDZIAŁ 1.	Pojęcie i własności algorytmu	9
	1.1. Przetwarzanie imperatywne	9
	1.2. Metody zapisu algorytmu	10
	1.3. Pseudokod	13
	1.4. Skończoność algorytmu	14
	1.5. Ogólny schemat konstruowania poprawnych algorytmów	14
ROZDZIAŁ 2.	Algorytmy iteracyjne i rekurencyjne	16
	2.1. Pętle iteracyjne. Warunek stopu	16
	2.2. Pętla for	17
	2.3. Przykłady algorytmów iteracyjnych	18
	2.4. Wyszukiwanie liniowe i binarne. Złożoność obliczeniowa algorytmów iteracyjnych	20
	2.5. Algorytmy rekurencyjne — pierwsze podejście	25
ROZDZIAŁ 3.	Typy danych proste i złożone	28
	3.1. Typy wartościowe i referencyjne	28
	3.2. Proste typy wartościowe	28
	3.3. Typy złożone — obiekty, struktury, tablice, słowniki	29
	3.3.1. Typ obiektowy i strukturowy	29
	3.4. Typ tablicowy. Tablice asocjacyjne (słowniki)	29
ROZDZIAŁ 4.	Algorytmy sortowania tablic	33
	4.1. Sortowanie przez proste wstawianie	34
	4.2. Sortowanie przez prostą zamianę (sortowanie bąbelkowe)	36
	4.3. Sortowanie szybkie (QuickSort). Metoda „dziel i zwyciężaj”	37
	4.4. Sortowanie z użyciem dodatkowej tablicy	38

ROZDZIAŁ 5.	Algorytmy i procesy rekurencyjne	40
5.1.	Anatomia przetwarzania rekurencyjnego	40
5.2.	Szacowanie złożoności obliczeniowej w rekurencji	42
5.3.	Derekursywacja	44
5.4.	Rekurencja ogonowa i bezogonowa	46
5.5.	Rekurencja zagnieżdżona	46
ROZDZIAŁ 6.	Programowanie liniowych struktur dynamicznych	48
6.1.	Cechy struktur dynamicznych	48
6.2.	Zjawiska na stosie i na sterwie w programowaniu struktur dynamicznych	49
6.3.	Oparte na referencji listy liniowe	51
6.3.1.	Lista liniowa jednokierunkowa	52
6.3.2.	Lista liniowa jednokierunkowa z wartownikiem	54
6.3.3.	Dynamiczne LIFO-stosy i FIFO-kolejki	56
6.3.4.	Samoorganizujące się listy	58
6.4.	Listy cykliczne	60
6.5.	Listy z przeskokami. Przeszukiwanie indeksowo-sekwencyjne	61
6.6.	Listy liniowe dwukierunkowe	62
ROZDZIAŁ 7.	Drzewa i lasy	64
7.1.	Rekurencyjna definicja drzewa	64
7.2.	Drzewa binarne	67
7.3.	Algorytm tzw. naturalnego przekształcenia dowolnego lasu w drzewo binarne	68
7.4.	Algorytmy przeglądania drzew binarnych	69
7.5.	Drzewa binarnych poszukiwań (drzewa BST)	71
7.6.	Drzewa wyważone i dokładnie wyważone	75
7.7.	Drzewa z priorytetem	77
ROZDZIAŁ 8.	Algorytmy obsługi grafów	79
8.1.	Grafy. Podstawowe pojęcia	79
8.2.	Metody reprezentacji grafu w pamięci	81
8.3.	Dynamiczna lista incydencji	83
8.4.	Rekurencyjny algorytm szukania w głąb dla grafu (algorytm DFS)	85

ROZDZIAŁ 9.	Algorytmy z nawrotami	90
9.1.	Ogólna postać algorytmu z nawrotami	90
9.2.	Klasyczne przykłady algorytmów z nawrotami	91
9.3.	Implementacje algorytmów z nawrotami	92
9.3.1.	Implementacja algorytmu z nawrotami oparta na zbiorach	93
9.3.2.	Implementacja algorytmu z nawrotami wykorzystująca drzewa poszukiwań	95
ROZDZIAŁ 10.	Metody usprawniania algorytmów o dużej złożoności czasowej	98
10.1.	Metody systematyczne	99
10.1.1.	Metoda obcinania gałęzi	99
10.1.2.	Metoda sklejanía gałęzi	100
10.1.3.	Metoda dekompozycji	100
10.2.	Metody heurystyczne	101
10.3.	Metody wykorzystujące sztuczną inteligencję	104
10.3.1.	Algorytm mrówkowy	104
10.3.2.	Algorytm genetyczny	105
ROZDZIAŁ 11.	Problemy algorytmicznie trudne	107
11.1.	Klasy problemów decyzyjnych	107
	Rozwiązania zadań ćwiczeniowych	109
	Bibliografia	121
	Skorowidz	123

ROZDZIAŁ 4. Algorytmy sortowania tablic

Sortowanie tablic to proces, którego wynikiem końcowym jest ustawienie elementów tablicy rosnąco lub w porządku odwrotnym — malejąco.

Opracowano wiele algorytmów fizycznego sortowania tablic, opartych na bardzo różnych pomysłach. Mówiąc o fizycznym sortowaniu, mamy na myśli algorytmy, które przemieszczają fizycznie elementy tablicy w pamięci komputera z jednego położenia w drugie w celu uzyskania uporządkowania. Jest to operacja bardzo kosztowna. Czas realizacji operacji przemieszczania elementów może być nawet kilka rzędów dłuższy niż innych operacji podstawowych, takich jak przykładowo przypisanie wartości do zmiennej.

Większość algorytmów sortowania fizycznego charakteryzuje się kosztem $O(N^2)$, gdzie N jest liczbą elementów tablicy. Algorytmy te wymagają bowiem dwóch pętli iteracyjnych, przy czym jedna z nich jest zanurzona w drugiej.

Przy ocenie jakości algorytmów sortowania (podobnie jak innych bardziej złożonych algorytmów) należy również brać pod uwagę ich prostotę i złożoność pamięciową.

Prostota algorytmu jest cechą dość istotną. Algorytmy o prostej strukturze, oparte na prostym pomysle, można bowiem łatwo modyfikować i dostosowywać do aktualnych potrzeb.

Kolejnym parametrem, który należy rozważyć, analizując własności algorytmów, jest zapotrzebowanie na dodatkową pamięć w trakcie pracy algorytmu. Tu algorytmy fizycznego sortowania nie są uciążliwe, na ogół bowiem sięgamy do metod tak zwanego **sortowania w miejscu**, nazywanych inaczej **metodami *in situ*** (łac.). Ich zapotrzebowanie na dodatkową pamięć ogranicza się zwykle do wielkości zajmowanej przez wartość pojedynczego elementu tablicy, przez który przesyłane są elementy tablicy celem ich zamiany.

W odniesieniu do algorytmów fizycznego sortowania, wybierając algorytm, należy zwrócić uwagę na wrażliwość na uporządkowanie sortowanej tablicy. Z tego punktu widzenia algorytmy sortowania dzielimy na:

- całkowicie niewrażliwe na uporządkowanie,
- częściowo wrażliwe na uporządkowanie,
- całkowicie wrażliwe na uporządkowanie.

W pierwszej grupie znajdują się algorytmy, dla których uporządkowanie tablicy (pierwotne bądź powstałe w trakcie sortowania) nie wpływa w sposób zasadniczy na czas realizacji algorytmu.

Za algorytmy częściowo wrażliwe na uporządkowanie uznamy te algorytmy sortowania, które w sposób znamieny ograniczają liczbę operacji wykonywanych w trakcie procesu sortowania (np. poprzez zawieszenie wykonywania pewnych pętli wewnętrznych).

Algorytmy sortowania całkowicie wrażliwe na uporządkowanie potrafią w trakcie realizacji algorytmu niejako przy okazji stwierdzić uporządkowanie tablicy (pierwotne bądź powstałe w dowolnym momencie procesu sortowania), przerywając natychmiast proces sortowania. Te algorytmy są oczywiście najefektywniejsze.

Algorytmy fizycznego sortowania tablic, jakkolwiek ciekawe z teoretycznego punktu widzenia, w chwili obecnej mają (głównie ze względu na swoją złożoność obliczeniową) znaczenie tylko historyczne, zostały bowiem zastąpione innymi metodami sortowania. Dwie z nich zostaną omówione w dalszej części.

Niżej omówiono działanie dwóch wybranych metod fizycznego sortowania tablic.

4.1. Sortowanie przez proste wstawianie

Pierwszy z dwóch wybranych algorytmów fizycznego sortowania tablic jest oparty na prostym pomysle zamiany dwóch elementów tablicy przez „przepompowanie” ich przez pojedynczą komórkę pamięci, jest to więc algorytm typu *in situ*. Algorytm jest całkowicie niewrażliwy

na uporządkowanie. Rysunek 4.1 pozwala zrozumieć ideę jego działania.

RYSUNEK 4.1.

Algorytm sortowania przez proste wstawianie:

a) stan wyjściowy;

b) stan

po zakończeniu

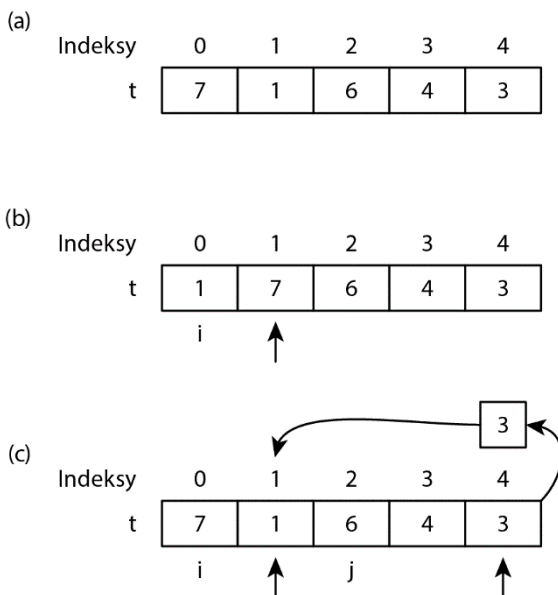
pierwszej iteracji;

c) ilustracja

procesu

przepisywania

elementów



Działanie algorytmu przebiega w dwóch pętlach: zewnętrznej i zagnieżdżonej w niej pętli wewnętrznej.

Pętla zewnętrzna przesuwaa indeks i wzdłuż tablicy. Dla każdego położenia indeksu i (rysunek 4.1b) pętla wewnętrzna przesuwaa w pozostałej części tablicy indeks j , wyszukując element najmniejszy w tej części tablicy. Jeśli go znajdzie, wymienia ten element z elementem $t[i]$ (rysunek 4.1c).

Złożoność obliczeniowa algorytmu sortowania przez proste wstawianie jest kwadratowa i wynosi $O(N^2)$, gdzie N jest rozmiarem zadania (liczbą elementów tablicy).

Spróbujmy udowodnić tę tezę.

W algorytmie sortowania przez proste wstawianie, którego ideę zaprezentowano na rysunku 4.1, w sytuacji najgorszego przypadku (gdy tablica jest odwrotnie posortowana) pętla zewnętrzna wykona się

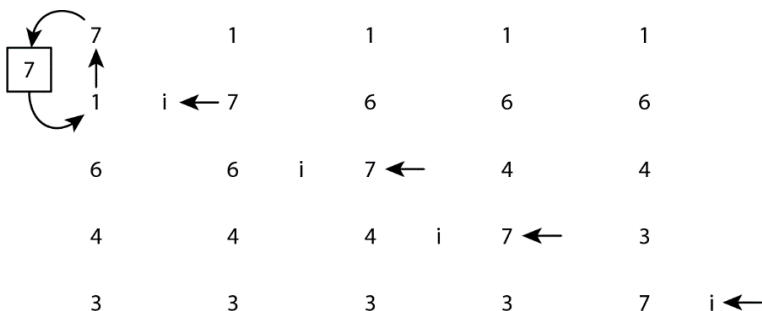
4 razy, a zanurzona w niej pętla wewnętrzna odpowiednio 3, 2 i 1 raz dla kolejnych położeń indeksu i . W sumie 6 razy.

Z powyższego wynika, że zanurzona w pętli wewnętrznej instrukcja zamiany elementów może wykonać się maksymalnie $4 \cdot 6 = 24$ razy, to jest nieco mniej niż podniesiona do kwadratu liczba elementów tablicy ($5^2 = 25$). Jak widać, jest to zgodne z definicją złożoności obliczeniowej (patrz podrozdział 2.4), która mówi o szacowaniu od góry (w tym przypadku przez N^2) zapotrzebowania na czas obliczeń w sytuacji najgorszego przypadku.

Algorytm uznamy za całkowicie niewrażliwy na uporządkowanie, ponieważ obie pętle wykonywać się będą niezależnie od początkowego uporządkowania elementów tablicy.

4.2. Sortowanie przez prostą zamianę (sortowanie bąbelkowe)

Drugim, bardzo popularnym algorytmem fizycznego sortowania jest algorytm sortowania przez prostą zamianę, znany jako sortowanie bąbelkowe. Rysunek 4.2 ułatwi zrozumienie działania tego algorytmu i jego własności. Jest to również algorytm typu *in situ*, całkowicie niewrażliwy na uporządkowanie, o złożoności obliczeniowej $O(N^2)$.



RYSUNEK 4.2. Algorytm sortowania bąbelkowego

W pierwszej kolumnie widzimy nieuporządkowaną tablicę ustawioną „pionowo”. Wzdłuż tablicy przesuwają się indeksy i , który sprawdza,

czy element poprzedzający element wskazywany przez indeks i jest większy od elementu wskazywanego przez i . Jeśli tak, dokonuje zamiany obu elementów, jak to widać na rysunku. W kolumnie drugiej widzimy efekt tej zamiany z indeksem i przesuniętym na kolejny element tablicy. Ostatnia kolumna przedstawia sytuację, w której indeks i doszedł w pętli do ostatniego elementu tablicy. Efektem tej operacji będzie „opadnięcie najcięższego bąbelka” (elementu o największej wartości) na dno, czyli do ostatniego elementu tablicy. Będzie się to wykonywać niezależnie od początkowego uporządkowania elementów.

Teraz całą opisaną wyżej operację trzeba powtórzyć, ale już dla nieco krótszego obszaru tablicy (nad pogrubioną linią). Musimy więc „opakować” pętlę, która przesuwawa indeks i , inną pętlą zewnętrzną, która będzie nadzorować wykonanie pętli wewnętrznej, dopóki długość tablicy dla pętli wewnętrznej nie skróci się do jednego elementu.

Ćwiczenie nr 9 do samodzielnego wykonania

Zapisz iteracyjny algorytm w postaci funkcji, która sortuje rosnąco przez prostą zamianę elementów tablicy przekazanej do funkcji poprzez parametr.

Ćwiczenie nr 10 do samodzielnego wykonania

Oceń i uzasadnij złożoność obliczeniową algorytmu sortowania bąbelkowego.

4.3. Sortowanie szybkie (QuickSort).

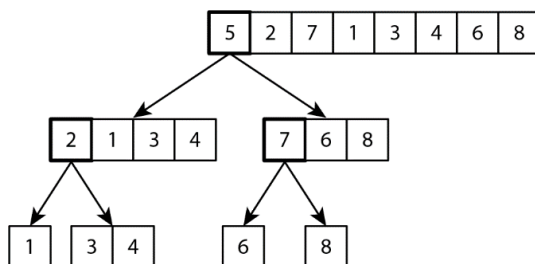
Metoda „dziel i zwyciężaj”

Jest to algorytm oparty na zupełnie innym pomysle w porównaniu z algorytmami omawianymi powyżej. Zastosowano tu (prezentowaną już przy okazji wyszukiwania binarnego) metodę dekompozycji problemu w wersji „dziel i zwyciężaj”. Algorytm jest algorytmem rekurencyjnym.

Wersja rekurencyjna tego algorytmu charakteryzuje się (co jest typowe dla algorytmów rekurencyjnych) niesłychaną prostotą. Składa się z dwóch kroków.

RYСУNEK 4.3.

Ilustracja idei
algorytmu
rekurencyjnego
sortowania
szybkiego
QuickSort



W pierwszym kroku na każdym poziomie rekurencyjnego wywołania algorytm rozdziela tylko elementy coraz krótszych tablic na dwie tablice: lewą, zawierającą elementy mniejsze od tzw. elementu osiowego, i prawą — zawierającą elementy większe od elementu osiowego. Elementem osiowym może być wybrany dowolny element. Tutaj rolę tę pełni element pierwszy każdej nowo powstałej tablicy.

Drugim krokiem algorytmu jest rekurencyjne wywołanie samego siebie dla każdej nowo powstałej tablicy. Po rozdzieleniu tablicy na pojedyncze elementy bądź dwuelementowe uporządkowane już tablice algorytm wycofuje się, dokonując na każdym poziomie łączenia dwóch swoich podtablic w jedną. Rekurencja bezogonowa zapewnia automatyczne połączenie tablic cząstkowych z odpowiednimi włączeniami elementów osiowych.

Napisano wiele odmian tego algorytmu. Ich złożoność obliczeniowa wynosi $O(N \cdot \log_2 N)$. Algorytm QuickSort jest więc algorytmem znacznie efektywniejszym czasowo niż omówione poprzednio dwa algorytmy fizycznego sortowania (patrz rysunek 2.10).

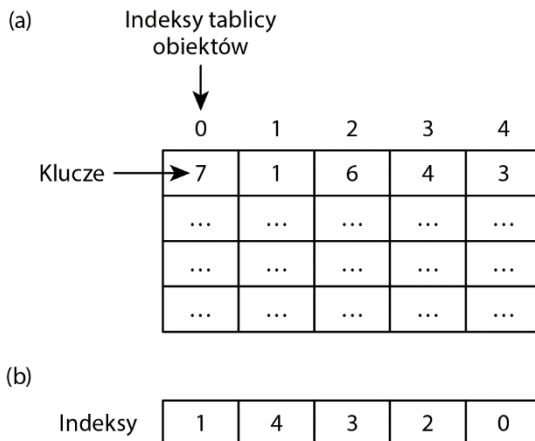
Niwelowana jest też zajętość pamięciowa, ponieważ algorytm ten nie musi dokonywać fizycznych przemieszczeń elementów tablicy, tworząc nowe podtablice. W zamian zmieniane są tylko referencje do elementów i podtablic.

4.4. Sortowanie z użyciem dodatkowej tablicy

Elementami tablicy są często rekordy bądź obiekty klas (na rysunku 4.4 w owalu), z których każdy zawiera szereg pól. Jedno z nich jest zwykle wybierane jako klucz sortowania.

RYSUNEK 4.4.

Tablica indeksowa
b) powstała
z posortowania
wejściowej tablicy
obiektów
a) wg wybranego
klucza



Zaprezentowana metoda sortowania sprowadza się do wytworzenia dodatkowej tablicy (zwanej **tablicą indeksową**), która przechowuje numery indeksów elementów tablicy oryginalnej, ale ustawione tak, że wybierając je kolejno, możemy według ich wartości wybierać obiekty z tablicy oryginalnej w kolejności rosnącego klucza.

W przytoczonym przykładzie pierwszy indeks tablicy indeksowej o wartości 1 pozwala wybrać obiekt tablicy oryginalnej o najmniejszej wartości klucza równej 1. Kolejny indeks z tablicy indeksowej o wartości 4 odpowiada obiektowi w tablicy oryginalnej o wartości klucza równej 3. Kolejne wartości indeksów tablicy indeksowej pozwolą wybierać kolejne obiekty z tablicy oryginalnej w porządku rosnącego klucza.

Metoda ta ma **minimalną złożoność pamięciową**, wymaga bowiem tylko użycia dodatkowej jednowymiarowej tablicy o rozmiarze tablicy oryginalnej, pozwalając zachować sortowaną tablicę w stanie pierwotnym. Wielką zaletą tej metody jest możliwość jednoczesnego generowania wielu tablic indeksowych wg różnych kluczy tablicy oryginalnej. Jednak jakakolwiek zmiana wartości kluczy tablicy oryginalnej pociąga za sobą konieczność generowania nowych tablic indeksowych.

Ćwiczenie nr 11 do samodzielnego wykonania

Zapisz w pseudokodzie iteracyjny algorytm tworzenia tablicy indeksowej dla klucza obiektów. Oceń czasową złożoność obliczeniową algorytmu.

Skorowidz

A

- adjacency matrix, *Patrz* macierz sąsiedztwa
- algorytm
 - definicja, 9
 - ewolucyjny, 105
 - fizycznego sortowania, 34
 - genetyczny, 105
 - imperatywny, 9
 - in situ, 34, 36
 - koszt, 22
 - mrówkowy, 104
 - ogonowy, 26, 40
 - prostota, 33
 - przekształcenia dowolnego lasu w drzewo binarne, 68
 - przeszukiwania
 - połówkowego, 23
 - quickSort, 38
 - rekurencyjny, 25, 37
 - skończoność, 14
 - szukania w głąb, 85
 - wielomianowy, 107
 - wycofywanie się, 27
 - wykładniczy, 107
 - z nawrotami, 90, 91
 - z powrotami, 86
 - zachłanny, 94, 103
- alokacja dynamiczna zmiennej, 50
- asercja
 - końcowa, 10
 - początkowa, 9

B

- bezpośredni następnik, 65
- poprzednik, 65
- Binary Search Tree, *Patrz* drzewo binarnych poszukiwań
- boolean, 28
- BST, *Patrz* drzewo binarnych poszukiwań

C

- char, 28
- ciąg liczb Fibonacciego, 46
- cykl, 80
- częściowy wektor rozwiązań, 94

D

- dane typu obiektowego, 29
- derekursywacja, 44, 46
- Document Object Model, *Patrz* model DOM
- dostęp sekwencyjny, 48
- double, 28
- dowiązanie, 65
- droga skoczka szachowego, 91
- drzewo
 - binarne, 67
 - binarnych poszukiwań, 72
 - definicja, 64
 - dokładnie wyważone, 76
 - drzewa i lasy, *Patrz* las drzew
 - głębokość, 65

gry, 104
korzeń, 65
liść, 65
poszukiwań, 95
stopień, 65
ścieżka, 66
uporządkowane, 65
wysokość, 65
wyważone, 76
z priorytetem, 77

E

ELEM, 86

F

feromon, 104
FIFO-kolejka, 55
first in — first out, 58
float, 28
frequency counter, 58
front, 58
funkcja
 Ackermanna, 47
 Fibonacciego, 43

G

garbage collector, 58
graf
 acykliczny, 80
 cykliczny, 80
 definicja, 79
 gęsty, 82
 nieskierowany, 80
 płaski, 80
 skierowany, 80
 spójność, 80
 ważony, 80
 zorientowany, 80

H

hard computing, 104
head, 56
heap, *Patrz* kopiec
HPO-drzewa, *Patrz* drzewa
 z priorytetem

I

inicjalizacja listy, 52
inorder, 70
int, 28
inteligencja grupowa, 104

K

key, 31
klasa, 29
klasa ELEM, 67
kopiec, 77, 78
koszt, 33

L

las, 66
las drzew, 48, 96
last in — first out, 58
liczba wywołań rekurencyjnych,
 43
licznik częstości, 59
LIFO-stosy, 56
lista
 incydencji, 83
 jednokierunkowa cykliczna,
 60
 krawędzi, 82
 liniowa, 48
 liniowa dwukierunkowa, 63
 samoorganizująca się, 58
 z przeskokami, 62

Ł

łuk, 65

M

macierz sąsiedztwa, 81
maksymalna zajętość pamięci, 43
mapy, 31
metoda
 dekompozycji problemu, 23, 37, 100
 dziel i zwyciężaj, 23, 37, 100
 in situ, 33
 klasy, 29
 MF, *Patrz* Move to Front
 obcinania gałęzi, 99
 porządku poprzedzającego, 70
 preorder, 69, 71, 96
 sklejania gałęzi, 100
 sortowania w miejscu, 33
 systematyczna, 44
 sztucznej inteligencji, 103
 zstępująca, 70
metody
 heurystyczne, 98, 101
 systematyczne, 98
 wykorzystujące sztuczną inteligencję, 98
model DOM, 81
Move to Front, 58

N

null, 50

O

obiekt, 29, 50
ogonek, 26

P

pętla
 do while, 16
 for, 17
 for each, 17
 iteracyjna, 17
 while, 16, 19, 53, 86
postorder, 70
problem
 decyzyjny, 107
 doboru, 91
 komiwojażera, 91, 101
 NP-zupełnych, 108
 ośmiu hetmanów, 91
 plecakowy, 92
proces iteracyjny, 16, 22
prosty typ wartościowy, 28
przepełnienie stosu, 47
przeszukiwanie
 binarne, 23
 dynamiczne, 104
 indeksowo-sekwencyjne, 62
 rekurencyjne, 25
pseudokod, 13

R

referencja pusta, 50
rekord, 29
rekurencja
 bezogonowa, 41
 definicja, 25
 głębokość, 42
 ogonowa, 46
 zagnieżdżona, 46
rekursja, *Patrz* rekurencja
return, 21
rozmiar zadania, 73

S

schemat blokowy, 11
self-organizing lists, *Patrz* lista
 samoorganizująca się
słowniki, 31
soft computing, 103
stack overflow, *Patrz*
 przepełnienie stosu
sterta, 51
stos, 50, 56
string, 18, 28
struktury dynamiczne, 48

Ś

ścieżka, 80

T

tablica
 asocjacyjna, 31
 dynamiczna, 31
 indeksowana, 20, 29, 39
 sortowanie, 33
tail, 56, 58
tail call, 46
trójargumentowa operacja
 warunkowa, 12
typ
 całkowity, 28
 logiczny, 28
 napisowy, 28
 referencyjny, 28, 49
 zmiennopozycyjny, 48
 znakowy, 28
typy danych
 proste, 28
 wartościowe, 28
 złożone, 28

U

uczenie się grupowe, 104

V

value, 31
VERT, 86

W

warunek stopu, 17
węzeł
 definicja, 65
 drzewa binarnego, 69
 wewnętrzny, 65
wyszukiwanie
 binarne, 22
 liniowe, 22
 warunkowe, 26
 wyczerpujące, 86, 90
wywołanie rekurencyjne, 25

Z

zapętlenie, 14
zawieszenie przetwarzania, 14
złożoność obliczeniowa
 czasowa, 22
 minimalna, 39
 pesymistyczna, 22
 wielomianowa, 43
zmienna
 lokalna, 18
 sterująca, 17
 typu referencyjnego, 50
 wskazująca, 50
 wskazywana, 50

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 

To, co najważniejsze w programowaniu

Algorytmy to skończone ciągi jasno zdefiniowanych czynności, prowadzących do zrealizowania określonych zadań. Ten podręcznik skupia się na algorytmach imperatywnych (od łac. *impero* — rozkazywać) i wprowadza podstawowe pojęcia algorytmiki niezbędne do nauki programowania. Uczy projektowania, zapisywania i analizy poprawności, jak również podstaw szacowania złożoności czasowej i pamięciowej algorytmów. Co więcej, zamieszczono tu wiele zadań, których rozwiązanie zaowocuje lepszym zrozumieniem teorii i pogłębieniem umiejętności praktycznych. By jeszcze bardziej ułatwić pracę z podręcznikiem, zawarta w nim treść została bogato zilustrowana rysunkami poglądowymi i fragmentami kodów.

Dzięki tej książce między innymi:

- Poznasz szereg ważnych algorytmów, jak wyszukiwanie binarne, sortowanie szybkie, algorytmy klasy *dziel i zwyciężaj*, algorytmy zachłanne itd.
- Nauczysz się korzystać z używanych powszechnie w programowaniu struktur danych: tablic, słowników, list wiązanych, stosów, kolejek, drzew binarnych i grafów
- Dowiesz się, jak stosować iterację i rekurencję w programowaniu
- Opanujesz podstawy języka Java

Helion 		KOD KORZYŚCI Sięgnij po więcej! ▶ 
 helion.pl		
 HELION SA ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	ISBN 978-83-283-9530-5  9 788328 395305	
Cena: 39,90 zł		