Filozofia Nauki
Rok XX, 2012, Nr 3(79)

Paweł Stacewicz

Co łączy umysł z teorią liczb?

Najbardziej naturalna — i rzecz jasna prawdziwa — odpowiedź na pytanie tytułowe artykułu brzmi: „Umysł łączy z teorią liczb fakt, że to umysł stworzył tę teorię i to umysł stosuje ją do rozwiązywania różnorodnych problemów".

W niniejszym tekście na odpowiedzi tej nie poprzestaniemy. Rozwiniemy natomiast jej drugą część, wskazując na *problem*, jaki umysł ma… ze samym sobą. Zapytamy zatem, czy umysł — który teorię liczb niewątpliwie rozwinął — potrafi sam siebie za jej pomocą opisać? A jeśli potrafi, to za pomocą jakiego jej fragmentu i przy użyciu jakiego typu liczb?

Z historycznego punktu widzenia problem nasz odnosi do starożytnego poglądu Pitagorejczyków, którzy głosili, że wszystko jest liczbą, zastanawiając się przy tym, jakie to liczby i jakie ich własności pasują najlepiej do poszczególnych cząstek wszystkiego. Jakie zatem pasują do umysłu? I co to znaczy — wyrażając nasz problem maksymalnie zwięźle — że „*umysł jest liczbą*"?[1]

1. KODOWANIE LICZBOWE

Jeśli odniesione do umysłu przypuszczenie pitagorejskie chcieć wyrazić w języku współczesnym, to trzeba wybić na pierwszy plan pojęcie *kodowania*. Pojęcie to znamy przede wszystkim z informatyki. Wszelkie informacje bowiem, o ile mają zostać przetworzone przez jakąś maszynę, muszą zostać zakodowane w postaci zrozu-

[1] Tekst ujęty w cudzysłów jest tytułem wystąpienia konferencyjnego, na którego podstawie powstał niniejszy artykuł. Zarówno wspomniane wystąpienie, jak i ten artykuł, są mocno powiązane z rozdziałem 10 książki *Umysł-Komputer-Świat. O zagadce umysłu z informatycznego punktu widzenia* (autorstwa W. Marciszewskiego i P. Stacewicza), gdzie niektóre z naszkicowanych tu wątków omówiono szerzej.

miałej dla tejże maszyny. Na przykład, aby sprzężony z kamerą komputer cyfrowy mógł dobrze interpretować docierające doń obrazy, muszą one zostać zapisane w postaci binarnej, czyli zero-jedynkowej. Podkreślić tu trzeba sprawę fundamentalną: otóż każdy komputer, niezależnie od wymaganej przezeń metody kodowania (binarnej, dziesiętnej, analogowej itp.), jest w istocie maszyną obliczeniową, to znaczy wszelkie przetwarzane w jego wnętrzu dane są (na najniższym poziomie reprezentacji) kodowane jako liczby, a operacje na nich są realizowane wewnątrz maszyny jako obliczenia. Wynika stąd, że najbardziej pierwotna cecha wewnątrzkomputerowego kodu to jego matematyczność czy też *liczbowość*.

> Oto prosta ilustracja powyższych stwierdzeń. Gdy na ekranie komputera śledzimy animację, np. jednostajny obrót pewnego kształtu, to w gruncie rzeczy we wnętrzu maszyny dokonują się super-szybkie *obliczenia*. Objaśnijmy je na poziomie nieco wyższym niż operacje binarne — poziomie działań na liczbach rzeczywistych.
>
> Załóżmy, że obracany kształt jest określony przez siatkę punktów, z których każdy ma trzy współrzędne: *x, y, z*. Aby uzyskać efekt obrotu, każda taka trójka jest mnożona przez tzw. macierz obrotu (czyli zadaną z góry tablicę liczb). Dzięki temu są wyznaczane nowe współrzędne i te nowe współrzędne określają położenie samochodu w kolejnej chwili. W owej kolejnej chwili punkty o starych współrzędnych znikają, pojawiają się zaś punkty o nowych współrzędnych. Pojawiają się po to, by po kolejnej operacji zbiorczego mnożenia ustąpić miejsca nowym punktom. A zatem: choć na ekranie widzimy ruch, to wewnątrz komputera dokonują się specjalnie dobrane operacje matematyczne.

Za efektywne przetwarzanie reprezentowanych liczbowo danych (takich choćby jak współrzędne punktów) odpowiadają, rzecz jasna, *programy komputerowe* (służące na przykład do mnożenia macierzy). To właśnie za ich sprawą idea kodowania znajduje najpełniejsze — rzec by można: dwustronne — rozwinięcie. Każdy program bowiem z jednej strony jest pewnym kodem (algorytmem zakodowanym w określonym języku programowania), a z drugiej strony to on właśnie steruje przetwarzaniem wewnątrz komputerowego kodu, czyli tak lub inaczej reprezentowanych informacji. Mówiąc krótko: *każdy program jest kodem, który służy do przetwarzania innych kodów*. A wyrażając to samo w konwencji teorioliczbowej: każdemu programowi odpowiada pewna liczba, będąca wynikiem zakodowania i scalenia w jeden zapis jego kolejnych instrukcji (np. liczba binarna); podobnie zaś danym przetwarzanym przez program-liczbę odpowiadają jakieś liczby.[2]

2. LICZBOWY KOD UMYSŁU?

Wiedząc na czym polega „liczbowy" charakter programów komputerowych, możemy przyjąć dalej — zgodnie z praktyką badawczą nauk kognitywnych — że umysł przypomina sztuczny system do przetwarzania informacji (krótko: komputer), a jako

[2] Przez liczbę odpowiadającą programowi można rozumieć ogromną liczbę binarną określoną cyfra po cyfrze przez binarny (tj. zero-jedynkowy) kod tego programu.

taki jest wyposażony w pewnego rodzaju oprogramowanie.[3] Owo oprogramowanie z kolei jest pewnym kodem, który można nazwać *kodem wewnątrzumysłowym*. Przy takich założeniach daje się wyrazić jasno, co znaczy neopitagorejskie przekonanie o tym, że „umysł jest liczbą".

Oto stosowne wyjaśnienie:

> *O ile istnieje wewnątrz-umysłowy kod, który na podobieństwo kodów sterujących pracą maszyn odpowiadałby za zachowanie ludzkiego organizmu, to kod ten daje się zapisać jako pewna liczba — liczba reprezentująca całość kodu i mieszcząca w sobie pełną informację o możliwych zachowaniach organizmu.*

Należy zwrócić uwagę na zawarty w powyższym zdaniu tryb warunkowy: o wewnątrzumysłowym kodzie nie twierdzimy kategorycznie, że istnieje; używamy natomiast frazy, że „o ile, istnieje to może mieć taki a taki charakter". Ów charakter z kolei określamy przez wstępne przypisanie umysłowi pewnych cech komputera. Strategię taką — nie przez wszystkich podzielaną — uzasadniają dość dobrze realne zastosowania komputerów, spośród których na pierwszy plan wysuwają się dwa.

Po pierwsze, tzw. zastosowania cybernetyczne (czyli sterownicze) pokazują, jak to jest możliwe, że programistyczne kody (*de facto* liczbowe) mogą *sterować* pracą maszyn, np. robotów przemysłowych, i w ten pośredni sposób oddziaływać na świat fizyczny. Skoro zaś równoważne pewnym liczbom programy komputerowe mogą to czynić, to powstaje uzasadniony domysł, że w podobny sposób umysł kieruje działaniem sprzężonych z nim układów biologicznych i organów. W ten sposób wnioskuje się o własnościach obiektu nie dość dobrze poznanego, czyli umysłu, na podstawie właściwości obiektu znanego doskonale, czyli komputera.

Po drugie jednak, za pomocą informatycznych kodów udaje się realizować — jeśli nie całkowicie, to w dość dobrym przybliżeniu — czynności wewnątrzumysłowe, a nie tylko nakierowane na świat zewnętrzny. Należą do nich, na przykład, wnioskowanie i uczenie się; a mówiąc ogólniej, wszelkie procesy poznawcze, które próbuje się modelować i/lub realizować komputerowo w ramach *badań nad sztuczną inteligencją*.[4]

Sumując: do uzasadnionej wiary w istnienie wewnątrzumysłowego kodu (liczbowego) skłania *informatyka*, a dokładniej rzecz biorąc, znajomość zasad działania programów komputerowych oraz ich „poznawczych" zastosowań.

---

[3] Założenie takie jest bardzo ogólne i nie wymaga określenia, o jaki komputer (np. cyfrowy, analogowy, kwantowy itp.) chodzi.

[4] Badania nad sztuczną inteligencją stanowią dobrze i szeroko rozwinięty dział współczesnej informatyki (por. [Russel, Norvig 1994]). Mogą poszczycić się też ogromną paletą realnych zastosowań, co motywuje dodatkowo do wiary w powodzenie strategii badawczej polegającej na porównywaniu umysłu z komputerem.

3. IDEA NIEOBLICZALNOŚCI

Narodziny informatyki — bez której idea umysłu-liczby nie mogłaby przemówić do wyobraźni człowieka współczesnego — zbiegły się w czasie z niezwykle ważnym odkryciem w teorii liczb. Chodzi o nowy podział liczbowego continuum na wielkości *obliczalne i nieobliczalne*. Dokonał go w roku 1936 Alan Turing, który w głośnej pracy *On computable numbers, with an application to Entsheidungsproblem* przedstawił koncepcję abstrakcyjnej maszyny, zwanej dziś *maszyną Turinga* [Turing 1936]. Maszyna ta, z jednej strony, stała się podstawą nowego podziału liczb, a z drugiej strony pozwoliła uściślić nie dość do tej pory jasne pojęcie algorytmu, czyli mechanicznej procedury przekształcania symbolicznych danych w symboliczne wyniki. Jak widać zatem, stała się matematycznym spoiwem informatyki i teorii liczb.

> Zakładając u czytelnika elementarną znajomość koncepcji Turinga, przypomnijmy drobnym drukiem, że maszynę, o której mowa, rozumie się dwojako. Z jednej strony, chodzi o pewne *urządzenie fizyczne* — wyposażone w podzieloną na komórki taśmę, głowicę do odczytu/zapisu danych oraz tablicę instrukcji — które to urządzenie odczytując, kasując i zapisując symbole na taśmie, a także zmieniając swoje stany wewnętrzne, realizuje program zawarty w tablicy instrukcji.
>
> Z drugiej strony, automat Turinga jest pewną *ideą teoretyczną*, która pokazuje, na czym polega mechaniczne przetwarzanie danych w wynik, czyli działanie algorytmiczne. Zgodnie z tąże ideą dane trzeba w sposób jednoznaczny zakodować, a potem poddać ściśle określonym przekształceniom, zestawionym jedno po drugim, bez żadnych luk, w programie maszyny. Przy takim rozumieniu fizyczna konstrukcja maszyny nie ma większego znaczenia; ważne, że działanie automatu jest jakoś skorelowane z jego programem. Turing podał po prostu pewien skrajnie prosty i raczej czysto teoretyczny schemat takiej korelacji (por. [Marciszewski, Stacewicz 2011, s. 120-124]).

Zwieńczeniem turingowskiej koncepcji automatu i algorytmu zarazem jest idea *maszyny uniwersalnej* — takiej maszyny, która potrafi symulować pracę dowolnego automatu konkretnego, a jako taka potrafi zrealizować każdy program.

Mówiąc bardzo zgrubnie, uniwersalna maszyna Turinga (UMT) działa w sposób następujący. Na jej taśmę wprowadza się zakodowany opis pewnej maszyny konkretnej M (chodzi głównie o definiujący ją program, czyli tablicę zmian stanów) oraz dane wejściowe maszyny M. Następnie maszyna UMT czyta naprzemiennie: raz dane wejściowe automatu M, raz jego tablicę instrukcji (tym właśnie steruje jej uniwersalny program) i zależnie od bieżących instrukcji wypisuje na taśmie odpowiednie symbole. Są to takie same symbole, jakie pojawiałyby się na taśmie maszyny M.[5]

Ze względu na jej nieograniczone możliwości symulacyjne maszynę UMT traktuje się jako ostateczną podstawę pojęcia *algorytmiczności*. Algorytmem bowiem wolno nazwać każdy schemat, który można zapisać jako program pewnej maszyny

[5] Przyrównując maszynę UMT do współczesnych komputerów cyfrowych, powiedzielibyśmy, że przypomina ona komputer wyposażony w system operacyjny, który to system — dzięki różnym kompilatorom, interpreterom itp. — zapewnia wykonanie każdego dostarczonego mu programu.

Turinga (konkretnej), a zatem taki program, który może wykonać uniwersalna maszyna UMT.[6]

I w tym właśnie miejscu dochodzimy do idei *nieobliczalności*, czyli niealgorytmizowalności. Prowadzą do niej następujące pytania: Czy istnieją problemy, których mimo precyzyjnej definicji nie sposób rozwiązać za pomocą maszyn Turinga? Czy istnieją zatem zagadnienia niemające rozwiązań w dziedzinie algorytmów dla maszyn cyfrowych?

Jak wiadomo, Alan Turing odkrył takie zagadnienie, uznawane dziś za kanoniczne, a nazywane powszechnie *problemem stopu*.

Oto jego sformułowanie w postaci pytania:

> *Czy istnieje taka maszyna Turinga MT, która dla dowolnej innej maszyny M_i (tj. jej zakodowanego opisu) oraz dla dowolnych jej danych wejściowych D_j zawsze potrafi rozstrzygnąć, czy maszyna M_i dla danych D_j zakończy pracę?*

Okazuje się, że maszyny takiej być nie może, co stanowi o niealgorytmizowalności, czyli nieobliczalności podanego problemu. Okazuje się nadto, że istnieje wiele innych problemów tego typu, których w całej ogólności, czyli dla wszelkich możliwych danych wejściowych, rozwiązać się nie da. Problemy takie zwie się w informatyce *nieobliczalnymi*. (Por. też [Harel 2000]).

4. NIEOBLICZALNOŚĆ W TEORII LICZB

Gdyby ustalenia Turinga co do nierozwiązywalności pewnych problemów zestawić z neopitagorejskim przekonaniem o tym, że wszystko jest liczbą, to trzeba by uznać, że istnieją *liczby, których nie można obliczyć*. Dlaczego?

Otóż konsekwentny zwolennik rzeczonego przekonania musi przyjąć, że każdemu problemowi i każdemu jego rozwiązaniu odpowiadają jakieś liczby. Z informatycznego punktu widzenia są to: komputerowy kod problemu oraz kod rozwiązującego go programu (równoważne pewnym liczbom). Od czasu Turinga wiadomo jednak, że istnieją problemy nierozwiązywalne za pomocą jakichkolwiek algorytmów. Wynika stąd, że ich niemożliwym do algorytmicznego uzyskania rozwiązaniom nie daje się przypisać żadnych, równoważnych kodom komputerowym, liczb. Skoro jednak, zgodnie z pitagorejskim przekonaniem, liczby takie istnieją, to muszą one należeć do kategorii *nieobliczalnych*, czyli niemożliwych do algorytmicznego wyznaczenia.

---

[6] Dopowiedzmy, że chodzi o algorytmiczność w kontekście technologii cyfrowej. Wynika to z faktu równoważności maszyn Turinga i maszyn cyfrowych. Niekiedy fakt ten wysławia się następująco: każdy program dla maszyny cyfrowej, niezależnie od jego stopnia złożoności i wymagań co do szybkości procesora, można odpowiednio zakodować i powierzyć do wykonania maszynie Turinga (zarówno konkretnej, jak i zdolnej do jej symulacji maszynie uniwersalnej).

Tak w istocie przedstawia się dostrzeżony przez Alana Turinga związek między sferą problemów nieobliczalnych i liczb nieobliczalnych. Dodajmy koniecznie, że związek ten zasadza się na koncepcji maszyny Turinga. To znaczy: i problemy, i liczby traktuje się jako nierozwiązywalne czy też niewyznaczalne za pomocą automatów Turinga właśnie.

Niezależnie od istnienia objaśnionego związku liczby nieobliczalne zostały określone pierwotnie jako pewien *podzbiór liczb rzeczywistych*. Ów podzbiór możemy nazwać klasą KLNO (klasa liczb nieobliczalnych), a rozłączny z nim podzbiór liczb obliczalnych — KLO (klasa liczb obliczalnych).

Chcąc opisać rozłączny podział KLO/KLNO ściśle, trzeba powołać się, rzecz jasna, na maszyny Turinga — zawężając jednak ich obszar zastosowań do czynności generowania symbolicznych zapisów liczb rzeczywistych, czyli ich kolejnych cyfr.

Oto stosowna charakterystyka:

> *Na klasę KLO składają się wszelkie możliwe wyniki obliczeń wszelkich możliwych maszyn Turinga dla wszelkich ich danych wejściowych. Mówiąc zaś inaczej: klasę KLO tworzą wszystkie liczby rzeczywiste, których symboliczne przedstawienia można generować algorytmicznie, z dowolną zadaną dokładnością.*
> *Klasę liczb nieobliczalnych KLNO wypełniają natomiast wszelkie wielkości pozostałe.*[7]

Przytoczone charakterystyki zabrzmią, być może, bardziej sugestywnie, jeśli skupimy uwagę na dziesiętnej reprezentacji liczb rzeczywistych. Przy takiej reprezentacji za liczby obliczalne trzeba uznać te wielkości, których kolejne cyfry przedstawienia dziesiętnego — zarówno przed kropką dziesiętną, jak i po niej — daje się uzyskiwać schematycznie, krok po kroku, zgodnie z pewną procedurą algorytmiczną. Zakłada się przy tym, że procedura ta, jeśli nie pozwala wyznaczyć danej liczby dokładnie, to pozwala przybliżyć ją z *dowolną zadaną dokładnością*.

Oto prosty przykład. Dla niewymiernej liczby Eulera *e* istnieje analityczny wzór ($e = \sum_{n=0}^{\infty} \frac{1}{n!} = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots$), który dla coraz większych *n* daje coraz lepsze przybliżenia wartości *e*, zbiegające granicznie do jednej jedynej wartości. Wzór ten stanowi odpowiednik algorytmu, który można zaimplementować w postaci programu, a to przesądza o obliczalności liczby *e*.

Podany przykład zwraca uwagę na pewien ważny fakt. Otóż pośród liczb niewymiernych — czyli takich wielkości, które mają nieskończone i nieregularne roz-

[7] O tym, że klasa KLNO jest niepusta, czyli faktycznie istnieją liczby inne niż obliczalne, przekonuje oryginalny dowód Alana Turinga — wzorowany na rozumowaniu przekątniowym G. Cantora, które przekonuje niezawodnie, że zbiór liczb rzeczywistych jest nieprzeliczalny. (Por. też [Marciszewski, Stacewicz 2011, s. 123]).

winięcia dziesiętne — znajdują się liczby dostępne obliczeniowo, a więc obliczalne (takie choćby jak *e*). Innymi słowy: zbiór KLO obejmuje nie tylko wielkości wymierne, ale również pewne szczególne wielkości niewymierne.[8]

Okazuje się natomiast, że klasa KLNO zawiera wyłącznie liczby niewymierne. Są to te spośród wielkości niewymiernych, których nie sposób przybliżać z dowolną zadaną dokładnością za pomocą jakichkolwiek maszyn Turinga (a więc jakichkolwiek programów dla maszyn cyfrowych). Co więcej liczb tych, to znaczy elementów klasy KLNO, jest nieprzeliczalnie wiele — w przeciwieństwie do wielkości obliczalnych, które jako całość są równoliczne ze zbiorem liczb naturalnych (a także wymiernych). (Por. [Marciszewski, Stacewicz 2011, s. 122-126]).

> Napotykając pojęcie liczby niemożliwej do obliczenia (nieobliczalnej), wyobraźnia ludzka ma pewien kłopot. Kłopot ten wiąże się z silnym przyzwyczajeniem do *symbolicznego zapisywania* liczb, najlepiej w postaci dziesiętnej. Dzięki zapisywaniu ludzie potrafią uchwycić liczby jako coś skończonego i zamkniętego w jednym napisie (jak w wypadku *0,75* na przykład) albo jako coś nieskończonego wprawdzie, lecz opisanego skończonym wzorem, pozwalającym nadto wyrażać liczbę coraz to dokładniej (jak w opisanym wyżej przypadku *e*).
>
> Jeśli teraz, mając na uwadze wskazany kontekst psychologiczny, spojrzymy na liczby nieobliczalne, to muszą się one wydać niezwykle dziwne. W ich wypadku bowiem nie mają zastosowania *żadne reguły zapisu* — gdyby reguły takie obowiązywały, to moglibyśmy sami lub za pomocą jakiejś maszyny podawać kolejne cyfry danej liczby, a to znaczyłoby, że jest ona obliczalna. Weźmy dla przykładu pewną wielkość nieobliczalną L, która jest „pokrewna i stosunkowo bliska" liczbie Eulera *e*. Jej dziwny, bo nieobliczalny, charakter moglibyśmy wyrazić jakoś tak: „*jest to liczba 2,718…, a potem nie wiadomo co, bez żadnej reguły, która pozwoliłaby nam kontynuować ten zapis*".
>
> Z uwagi na frazę „*nie wiadomo co*" liczby nieobliczalne można postrzegać jako liczby całkowicie losowe — to znaczy takie liczby, w których przedstawieniu dziesiętnym, począwszy od pewnej cyfry tego przedstawienia, układ kolejnych cyfr (jeśli nie wszystkich, to przynajmniej niektórych) jest całkowicie niezdeterminowany. Jeśli zaś jest niezdeterminowany, to nie może istnieć reguła pozwalająca określać te kolejne cyfry.

Ponieważ opisany wyżej podział liczbowego continuum na wielkości obliczalne i nieobliczalne stanowi punkt wyjścia do dalszych analiz (dotyczących już umysłu), warto podsumować własności obydwu typów liczb. Oto stosowna tabelka.

---

[8] Klasa niewymiernych liczb obliczalnych, czyli dość wyjątkowych liczb niewymiernych, jest mocno zróżnicowana. Należą do niej przede wszystkim niewymierne liczby algebraiczne, czyli takie które są pierwiastkami wielomianów o wymiernych współczynnikach (np. liczba $\sqrt{2}$, która jest pierwiastkiem wielomianu $x^2 - 2$), ale także pewne ważne liczby nie-algebraiczne (tzw. liczby przestępne, jak π czy e).

Liczby obliczalne	Liczby nieobliczalne
1a. *Istnieją maszyny Turinga zdolne obliczać je z dowolną dokładnością (krótko: są obliczalne algorytmicznie).*	**1b.** *Nie istnieją maszyny Turinga zdolne obliczać je z dowolną dokładnością (krótko: nie są obliczalne algorytmicznie).*
2a. *Jest ich przeliczalnie wiele (tj. tyle co liczb naturalnych i wymiernych).*	**2b.** *Jest ich nieprzeliczalnie wiele (tj. istotnie więcej niż liczb naturalnych).*
3a. *Są wśród nich liczby niewymierne.*	**3b.** *Wszystkie są niewymierne.*

5. NIEOBLICZALNOŚĆ W TEORII UMYSŁU

Znając turingowskie rozróżnienie między wielkościami obliczalnymi i nieobliczalnymi (które ma za podstawę ideę algorytmiczności), wiedząc nadto, że umysł wolno opisywać w kategoriach liczbowych (przyrównując sam umysł do komputera, a jego informacyjną zawartość do komputerowego oprogramowania), możemy postawić dwie hipotezy, tyczące się wewnątrzumysłowego kodu.

Oto hipoteza H1:

> *Istnieje obliczalny kod umysłu, który mógłby zostać — ze względu na swoją obliczalność właśnie — przekształcony w kod pewnej maszyny Turinga.*
> Mówiąc krótko: *każdy indywidualny umysł jest jakąś maszyną Turinga.*

Oto hipoteza H2:

> *Nie istnieje obliczalny kod umysłu, to znaczy kod, który można by przełożyć — bez utraty jego obliczeniowej mocy — na program pewnej maszyny Turinga.*
> Mówiąc krótko: *umysł nie jest żadną maszyną Turinga.*

Przedstawiając obydwie hipotezy w wersji skrótowej i hasłowej, nawiązującej swoim brzmieniem do eksponowanego w tym tekście motta Pitagorejczyków, moglibyśmy powiedzieć tak:

> hipoteza H1 głosi, że *umysł jest liczbą obliczalną*;
>
> hipoteza H2 natomiast, że *umysł jest liczbą nieobliczalną*.

Sformułowane przypuszczenia są wysoce spekulatywne, a ponadto odnoszące się do matematycznej teorii liczb, co na pierwszy rzut oka utrudnia myślenie o nich w kategoriach empirycznej sprawdzalności. Pierwsze wrażenie może być jednak mylące. Przypuszczenia nasze wiążą się bowiem nie tylko z formalną teorią liczb, ale również z *informatyką* — a to za sprawą objaśnionej wyżej równoważności liczb maszynom Turinga, a maszyn Turinga komputerom cyfrowym. Z tego względu można je częściowo weryfikować, to znaczy sprawdzać, jak dobrze różne programy dla ma-

szyn cyfrowych (głównie te spod znaku sztucznej inteligencji) służą sztucznej realizacji pewnych czynności poznawczych.

Do spraw tych powrócimy w kolejnym punkcie, w tym miejscu natomiast musimy rozjaśnić dwie kwestie związane z brzmieniem hipotez H1 i H2, które w pewnym uproszczeniu przyjmują postać quasi-matematycznej równości „*umysł = liczba (obliczalna bądź nie)*".

Kwestia pierwsza tyczy się lewej strony równości, czyli pojęcia umysłu. Chociaż piszemy ogólnie „umysł", to w domyśle mamy nie jakiś abstrakcyjny <umysł>, reprezentujący ogół różnych umysłów ludzkich, lecz każdy *indywidualny* umysł ludzki. Zgodnie z taką interpretacją każdemu indywidualnemu umysłowi U_i przysługuje odrębny kod $K(U_i)$, który reprezentuje odrębna liczba $L(U_i)$ (obliczalna bądź nie).

Przy tego rodzaju wykładni obydwie hipotezy pociągają za sobą tezę o *nieskończonej różnorodności* indywidualnych umysłów (a być może też tezę o ich potencjalnie nieskończonej złożoności). Wynika to z faktu, że zarówno liczby obliczalne, jak i nieobliczalne, są nieskończenie liczne (choć tych pierwszych jest mniej niż drugich). A zatem: niezależnie od tego, czy przyjmiemy H1 czy H2, to liczby kodujące umysły mogą być dowolnie duże, a same kody dowolnie długie. Skoro zaś w dziedzinie potencjalnych kodów występuje taka różnorodność, to najprawdopodobniej występuje ona także w dziedzinie obiektów kodowanych, czyli umysłów. Jeśli uznamy nadto, że długość kodu stanowi wiarygodny miernik złożoności umysłu, to dobrze uzasadniona wydaje się myśl o potencjalnie nieskończonej *złożoności* indywidualnych umysłów.

Kwestia druga tyczy się prawej strony równości, czyli pojęcia liczby — rozumianej bądź jako wielkość obliczalna, bądź jako nieobliczalna. Choć dwa wskazane atrybuty traktuje się w teorii liczb jako przeciwstawne (co znaczy tyle, że generują one rozłączny i wyczerpujący podział continuum liczb rzeczywistych na wielkości dwojakiego rodzaju), to w kontekście teorii umysłu trzeba rozumieć je *komplementarnie*. Nieobliczalność umysłu jako całości nie wyklucza zatem obliczalności pewnych zjawisk czy procesów mentalnych. Znaczy to, iż mówiąc „nie istnieje obliczalny kod umysłu", dopuszczamy możliwość istnienia obliczalnych fragmentów tego kodu. Fragmenty takie opisują te czynności poznawcze (wyizolowane spośród innych), które da się powierzyć maszynom Turinga.

Założenie takie nie powinno budzić wątpliwości. Wszak ludzki umysł potrafi działać algorytmicznie (czyli tak jak maszyna Turinga), a niektóre jego funkcje (choćby pewne formy wnioskowań sformalizowanych) udało się zalgorytmizować. Być może jednak istnieją zjawiska i procesy (choćby świadomość), których istotę oddają liczby nieobliczalne — jeśli zatem chcielibyśmy dołączyć ich opis do całościowego kodu umysłu, to kod ten stałby się nieobliczalny, a to wskutek nieobliczalności pewnych swoich składników. Reasumując zatem: hipotetyczny, nieobliczalny kod umysłu mieści w sobie obliczalne sub-kody, ale jako całość jest on czymś więcej niż suma owych sub-kodów, to znaczy odzwierciedla większą poznawczą i praktyczną moc umysłu niż suma tychże sub-kodów.

6. OBLICZALNOŚĆ UMYSŁU A JEGO REALIZACJA KOMPUTEROWA

Ponieważ odnoszące do teorii liczb, a dotyczące wewnątrzumysłowego kodu, hipotezy H1 i H2 daje się wyrazić w stylistyce informatycznej (wybijając na pierwszy plan pojęcie maszyny Turinga), to są one w istocie pewnego rodzaju odpowiedziami na pytanie o *komputerową realizację* umysłu. Stwierdzenie to rozwiniemy dla każdej hipotezy z osobna.

Zważywszy na wcześniejsze objaśnienia, tezę pierwszą można zapisać na trzy równoważne sposoby:

(H1a): Kod indywidualnego umysłu $K(U_i)$ wyraża się liczbą obliczalną $L(U_i)$.

(H1b): Kod indywidualnego umysłu $K(U_i)$ jest kodem jakiejś maszyny Turinga MT_i.

(H1c): Kod indywidualnego umysłu $K(U_i)$ jest jakimś programem dla maszyny cyfrowej.

Zapis (H1c) przesądza o tym, że w myśl hipotezy pierwszej *umysł jest realizowalny komputerowo*, a odpowiednich środków po temu dostarczyć może technologia, którą już dysponujemy, czyli sprowadzalna do maszyn Turinga technologia *cyfrowa*. Idzie, rzecz jasna, o realizację czysto hipotetyczną — bo każdy zgodzi się z tym, że nikomu jeszcze, z uwagi na kolosalną złożoność tego zadania, nie udało się zalgorytmizować wszelkich czynności poznawczych.

Co więcej, na całą sprawę należy patrzyć nie statycznie, lecz *dynamicznie* (por. [Stacewicz 2010, s. 209-211]). Wiadomo przecież, że umysły ludzkie, między innymi za sprawą wspomagających je maszyn, nieustannie się rozwijają. Rozwój ten skutkuje, po pierwsze, stopniowym wzrostem stopnia złożoności poszczególnych intelektów, a po drugie, poszerzaniem zakresu problemów przez nie rozwiązywalnych. Sytuacja taka wymaga, by programy komputerowe były w stanie „podążać" za rozwijającymi się umysłami i podobnie do nich zwiększać stopniowo swój stopień złożoności oraz swoją obliczeniową moc. Co najważniejsze jednak: zgodnie z hipotezą H1 jest to *wykonalne*.

Podkreślić trzeba jeszcze jedną rzecz. Ze względu na nieskończoną liczność zbioru liczb obliczalnych, można mówić nie tylko o sztucznych realizacjach umysłów ludzkich (tj. programach dla maszyn cyfrowych zdolnych symulować działanie konkretnych umysłów z dowolną dokładnością), ale również o sztucznych *kreacjach* umysłów podobnych, choć maszynowych. Takim bytom wykreowanym odpowiadałyby liczby nieprzysługujące żadnym umysłom istniejącym, choć odpowiednio mocno złożone (na miarę złożoności umysłów ludzkich). I w tym przypadku jednak, podobnie jak w opisanym wyżej przypadku sztucznych realizacji, interpretowana dynamicznie hipoteza H1 przewiduje jak najbardziej wykonalny scenariusz „podążania" umysłów maszynowych za rosnącą mocą obliczeniową umysłów zwykłych.

Popuszczając nieco wodze fantazji, można sobie wyobrazić sytuację następującą: umysły ludzkie rozwijają się (np. poprzez ewolucję), zyskując w kolejnych pokoleniach coraz większą złożoność (jej granica prawdopodobnie nie istnieje); podobnie jednak mogą rozwijać się umysłopodobne maszyny, które za sprawą coraz większej złożoności technologicznej (coraz dalej idąca miniaturyzacja, coraz większe pojemności pamięci itd.) oraz programistycznej (coraz bardziej skomplikowane kody programów) nadążają za rozwijającymi się umysłami.

Co więcej też, ponieważ obydwa równoległe światy, świat umysłów i świat maszyn, nie mogą przekroczyć granic obliczalności, to nie istnieją problemy, które byłyby nierozwiązywalne dla umysłów maszynowych, a byłyby rozwiązywalne dla umysłów ludzkich. Mówiąc inaczej: obliczalne umysły i sprowadzalne do maszyn Turinga automaty są sobie pokrewne również pod względem swoich ograniczeń.

7. NIEOBLICZALNOŚĆ UMYSŁU A JEGO REALIZACJA KOMPUTEROWA

Po przedstawieniu różnych „komputerowych" implikacji tezy H1, przejdźmy teraz do hipotezy H2, która zrównuje kod umysłu z jakąś liczbą nieobliczalną.

Podobnie jak w punkcie 6 możemy zaproponować trzy równoważne postaci tej tezy.

(H2a): Całościowy kod indywidualnego umysłu $K(U_i)$ wyraża się liczbą nieobliczalną $L'(U_i)$.

(H2b): Całościowy kod indywidualnego umysłu $K(U_i)$ nie jest kodem żadnej maszyny Turinga MT_i.

(H2c): Całościowy kod indywidualnego umysłu $K(U_i)$ nie może zostać przedstawiony jako kod programu dla maszyny cyfrowej.

Tym razem zapis trzeci przekreśla możliwość pełnej cyfrowej realizacji umysłu; nie wyklucza jednak możliwości innych, związanych z technikami, które uznaje się za *alternatywne* w stosunku do cyfrowych.

Czysto negatywne brzmienie hipotezy H2 („kod umysłu jakiś nie jest", to znaczy „nie wyraża się żadną liczbą obliczalną") sprawia, że nie daje ona żadnych wskazówek co do tego, o jakie konkretnie techniki alternatywne miałoby chodzić. Hipoteza H2 ukazuje zatem szeroki obszar *niewiedzy*. Próbując go zawęzić, trzeba stawiać kolejne hipotezy cząstkowe — dotyczące tego, czym kod umysłu mógłby być — i rozstrzygać je w odniesieniu do nowych, teoretycznych i praktycznych, dokonań informatyki.

Spośród dokonań tych na szczególną uwagę zasługują takie metody przetwarzania danych, które angażują *elementy losowe*. Na ich przydatność zdaje się wskazywać interpretacja kodujących umysł liczb nieobliczalnych jako wielkości częściowo przynajmniej losowych (zob. pkt 4). Ponieważ za ich dokładny kształt odpowiada wymykający się jakimkolwiek regułom przypadek, to być może tenże przypadek jest niezbędny, aby zrealizować nieobliczalny umysł komputerowo. A mówiąc inaczej:

być może programy komputerowe działające i/lub modyfikowane w sposób losowy lepiej oddają istotę umysłu niż schematy deterministyczne (por. [Marciszewski, Stacewicz 2011, s. 81]. Być może też taka właśnie, niedeterministyczna, czyli losowa, droga wiedzie ku maszynowej realizacji *inwencji* (dopowiedzmy, że w wypadku umysłu ludzkiego to inwencja właśnie decyduje o jego twórczej nieprzewidywalności).

Nie chcąc pozostać na poziomie stwierdzeń tak ogólnych, omówimy bardziej szczegółowo dwie metody przetwarzania danych, które, po pierwsze, angażują elementy losowe, a po drugie, są uznawane za alternatywne w stosunku do metod cyfrowych. Chodzi o techniki *kwantowe* i *ewolucyjne*.

Losowość pierwszych ma charakter zdecydowanie sprzętowy, można powiedzieć nawet: fizykalny. Oto na najniższym poziomie przetwarzania, poziomie kwantowych bramek logicznych, wykorzystuje się naturalne dla mikroświata (np. dla cząstek elementarnych) zjawiska *niedeterministyczne*. Wskutek ich istnienia wyniki pojedynczych operacji komputera kwantowego, a właściwie ich odczyty, są przypadkowe; dopiero wykonanie wielu takich operacji oraz uśrednienie odczytów daje wynik pożądany ze względu na realizowany cel. Ów wynik jednak powstaje w efekcie kumulacji szeregu zdarzeń losowych, a to nasuwa myśl, że można starać się tak ukierunkować przetwarzanie, by uzyskiwać wyniki nie tyle pożądane czy oczekiwane, ile oryginalne lub zaskakujące (w odniesieniu do umysłu ludzkiego nazwalibyśmy je pomysłowymi).

Druga ze wskazanych technik obliczeniowych, technika ewolucyjna, nawiązuje także do przyrody, choć tym razem ożywionej. Jak wiadomo bowiem, w świecie istot żywych zachodzi *naturalna ewolucja* — wsparta na takich procesach jak przypadkowe mutacje i krzyżówki, oraz po części losowa procedura selekcji. I właśnie te procesy, a właściwie ich maszynowe odpowiedniki, próbuje się wykorzystywać w przetwarzaniu danych. Zamiast przekształcać dane sekwencyjnie, dążąc do wyniku zgodnie z jakimś ściśle określonym schematem deterministycznym, generuje się całe populacje wyników próbnych, a następnie poddaje się je sztucznej ewolucji — oczekując, że w jej efekcie zostanie wygenerowany wynik optymalny.

Strategię taką cechuje *ukierunkowany indeterminizm*, a mówiąc bardziej technicznie, ukierunkowany globalnie indeterminizm lokalny. Określenia te wyrażają sytuację następującą. Lokalnie, a więc na poziomie takich mikrooperacji, jak mutacje i krzyżówki, działa czysty przypadek; natomiast globalnie, na poziomie makro, czyli przy ocenie i promowaniu do kolejnych populacji pewnych próbnych rozwiązań, działa kierunkująca selekcja (w jej efekcie z większym prawdopodobieństwem są promowane rozwiązania lepsze, a z mniejszym — gorsze). Co najważniejsze jednak, dokładny wynik symulowanej ewolucji nigdy nie jest przesądzony z góry, a reguły kierujące ewolucją są regułami niedeterministycznymi.[9]

[9] Więcej szczegółowych informacji o technikach ewolucyjnych w informatyce zawierają pozycje [Michalewicz 1992] oraz [Goldberg 1998].

8. PRZYKŁADY PODOBNYCH ZAGADNIEŃ

W dotychczasowym tekście poruszyliśmy tyko jedno zagadnienie filozoficzne — związane z turingowskim podziałem continuum liczbowego na wielkości obliczalne i nieobliczalne. Ze względu na kluczową dla Turinga koncepcję maszyny uwagi nasze koncentrowały się wokół perspektyw komputerowej realizacji czynności umysłowych.

Pozostając w kręgu tych samych przemyśleń, na zasadzie niewielkiego dopowiedzenia jednak, opiszemy teraz dwa inne zagadnienia, odwołujące się do innych nieco własności liczb rzeczywistych niż te, które dostrzegł i zbadał Alan Turing.

Zagadnienie pierwsze stanowi w pewnym sensie nieco inny, uproszczony i historycznie wcześniejszy, wariant kwestii omawianej w poprzednich punktach. Wysunął je Leibniz, a zinterpretował w duchu informatycznym Witold Marciszewski.[10] Odwołuje się ono do znanego już Pitagorejczykom rozróżnienia między wielkościami *wymiernymi* i *niewymiernymi*, które to rozróżnienie przypomina dystynkcję obliczalne/nieobliczalne, ale nie jest z nią tożsame (niektóre liczby niewymierne cechuje bowiem obliczalność, a inne nieobliczalność).

Mając na oku wskazane rozróżnienie, argumentuje się, że jeśli kod umysłu $K(U_i)$ wyraża się liczbą niewymierną $L(U_i)$, to z uwagi na nieskończone i nieregularne przedstawienie dziesiętne tejże liczby kodu $K(U_i)$ nie można poznać i obliczyć dokładnie, lecz tylko w przybliżeniu (podobnie jak nie sposób podać całej postaci dziesiętnej liczby $L(U_i)$). Dlatego też umysł opisany liczbą niewymierną mógłby być realizowalny za pomocą komputerów cyfrowych, ale tylko w *przybliżeniu*. Trzeba pamiętać przy tym, że Leibniz nie wiedział jeszcze o istnieniu obliczalnych liczb niewymiernych, nie znał też ścisłego pojęcia algorytmu (jako schematu wykonalnego przez maszynę Turinga); stąd też jego domysł, choć przenikliwy, musiał pozostać dość ogólny.

Zagadnienie drugie odwołuje się do innej jeszcze właściwości zbioru liczb rzeczywistych: otóż zbiór ten jest *ciągły*, to znaczy nie zawiera żadnych liczbowych luk.[11] Mimo to w rzeczywistoliczbowym continuum można wskazywać pewne nieskończone podzbiory, którym nie przysługuje już atrybut ciągłości. Najprostszy z nich to „dziurawe z natury" liczby naturalne, inny to liczby wymierne, jeszcze inny to wielkości obliczalne. Jak widać zatem, występuje w teorii liczb ważna dychotomia *ciągłe/nieciągłe*, którą uzyskuje się, „wyłuskując" z ciągłego zbioru liczb rzeczywistych pewne nieciągłe, choć nieskończenie liczne, podzbiory. (Por. [Mioduszewski 1996]).

---

[10] Zagadnienie to jest omówione szerzej w książce Witolda Marciszewskiego *Sztuczna inteligencja* (s. 23-29). Oznaczono je tam jako L_M, co sam pomysłodawca tłumaczy jako <Leibniz zinterpretowany przez Marciszewskiego>.

[11] Dopowiedzmy, że za ciągłość tę płaci się cenę nieobliczalności, bo luki między obliczalnymi liczbami rzeczywistymi wypełniają wielkości nieobliczalne. To zaś wiąże obecne zagadnienie z problemem nieobliczalności.

W informatyce natomiast dychotomii tej odpowiada ważka różnica między *analogowymi* i *cyfrowymi* technikami przetwarzania danych. Wyjaśnijmy krótko: używane najpowszechniej metody cyfrowe służą do obróbki danych zapisanych za pomocą skończonego zestawu symboli, np. cyfr (najczęściej zer i jedynek); mniej popularne metody analogowe pozwalają natomiast przetwarzać dane ciągłe, niemożliwe do pełnego opisu symbolicznego, a reprezentowane fizykalnie jako natężenia (potencjalnie ciągłe) pewnych wielkości fizycznych, np. prądu.

I tutaj właśnie, przy przejściu od teorii liczb do informatycznych zastosowań, zjawia się nasz filozoficzno-inżynierski dylemat. Które metody zapewniają większą skuteczność maszynowej realizacji czynności poznawczych? *Analogowe* czy *cyfrowe*?

Za przewagą pierwszych zdaje się przemawiać teoria — czyli fakt, że odpowiadające analogowości liczby rzeczywiste są liczniejsze od nieciągłych liczb wymiernych (a nawet obliczalnych) i pokrywają całe continuum możliwych do przetworzenia sygnałów. Teoretycznie zatem — mimo olbrzymiej skuteczności technik cyfrowych — metody analogowe dają „coś więcej", bo nie wiąże ich ograniczenie nieciągłości danych.

Tu jednak zjawia się wątpliwość. Być może bowiem ciągłość jest swego rodzaju matematyczną *fikcją*, która w pewnych sytuacjach zapewnia elegancki symboliczny opis obiektów wyidealizowanych (przejawem owej elegancji jest, na przykład, rachunek różniczkowo-całkowy). Być może w świecie komputerów — z ich obliczeniową zdolnością do bardzo dokładnego przybliżania wielkości ciągłych za pomocą wielkości dyskretnych — fikcja ta jest zbędna. Są to pytania i wątpliwości zarazem, na które nauka nie daje ostatecznej odpowiedzi.

9. PODSUMOWANIE: TEORIA UMYSŁU A TEORIA LICZB

Po rozwinięciu składających się na ten tekst punktów 1-8 jesteśmy gotowi do odpowiedzi na pytanie tytułowe, o to, co łączy teorię umysłu z teorią liczb. Otóż obydwie dziedziny spaja *informatyka*, a czyni to za sprawą pojęcia kodowania, które stosuje się — co jest bardzo ważne — do umysłu porównywanego z komputerem.

Dzięki pojęciu kodowania zawartość informacyjną umysłu (swoiste mentalne oprogramowanie), można utożsamić z pewnym *kodem* — z punktu widzenia informatyki chodzi o zbiór precyzyjnych *instrukcji*, możliwych do zapisania w pewnym programistycznym języku; z punktu widzenia matematyki chodzi natomiast o *kod liczbowy*, będący wynikiem odwzorowania w liczbach kolejnych instrukcji wspomnianego języka, a następnie scalenia tych instrukcji w jedną liczbę. Na pewnym, odpowiednio wysokim, poziomie abstrakcji ma zatem sens utożsamienie umysłu z liczbą.

Jeśli przyjmie się takie uzasadnienie i założy się zapośredniczoną w informatyce odpowiedniość między sferą liczb a światem zjawisk mentalnych, to uzyskuje się bardzo ciekawą i potencjalnie twórczą możliwość *heurezy*. Innymi słowy: zyskuje się

sposobność do wnioskowania z tego, co wiadome o liczbach, o tym, czego chcemy się dowiedzieć o umyśle. Wspomniana heureza może prowadzić od określonych własności i typów liczb (wymiernych, niewymiernych, obliczalnych, nieobliczalnych itp.) do hipotez o umyśle (np. takich, że kod umysłu wyraża się liczbą określonego rodzaju).

Niniejszy tekst stanowi niewielką próbkę strategii heurystycznej, wiodącej od matematycznych pojęć obliczalności i nieobliczalności, przez informatyczne rozróżnienie między cyfrowymi i niecyfrowymi technikami przetwarzania danych, aż do filozoficznego pytania o to, czy kod umysłu wyraża się liczbą obliczalną, czy też nieobliczalną.

Inspirowana teorią liczb konkluzja tekstu jest następująca. Modelując i/lub realizując umysł za pomocą informatycznych technik obliczeniowych, trzeba wziąć pod uwagę, że w teorii liczb istnieją wielkości *nieobliczalne*. Trzeba je traktować, po pierwsze, jako wielkości o nieregularnym i losowym (przynajmniej częściowo) przedstawieniu dziesiętnym, a po drugie, jako takie wielkości, które dołączone zbiorczo do klasy liczb obliczalnych zapewniają ciągłość zbioru liczb rzeczywistych. Zgodnie z podaną charakterystyką wielkości tych nie można obliczyć dokładnie, tj. z dowolną zadaną dokładnością, za pomocą żadnej maszyny Turinga.

O ile uznać zatem, że kod umysłu wyraża się liczbą nieobliczalną, a jego nieobliczalne składniki są istotne dla sprawności poznawczej człowieka, to trzeba zgodzić się, że dla komputerowej realizacji umysłu *nie wystarczą* równoważne maszynom Turinga techniki cyfrowe. Szukając zaś technik alternatywnych, warto mieć na uwadze wspomniane wyżej atrybuty nieobliczalności: losowość oraz gwarancję ciągłości liczbowego continuum. Jest to ważna wskazówka heurystyczna. Kierując się nią, warto poszukiwać takich rozwiązań, które łączą w sobie *niedeterministyczne* schematy przetwarzania danych (np. ewolucyjne) z *analogowością* przetwarzanych sygnałów i/lub wykonywanych operacji.

BIBLIOGRAFIA

Fichtenholz G.M. (1997), *Rachunek różniczkowy i całkowy (tom 1)*, Warszawa, Wydawnictwo Naukowe PWN.

Goldberg D.E. (1998), *Algorytmy genetyczne i ich zastosowania*, przeł. K. Grygiel, Warszawa, Wydawnictwa Naukowo-Techniczne.

Harel D. (2000), *Rzecz o istocie informatyki. Algorytmika*, przeł. Z. Weiss, P. Carlson, Warszawa, Wydawnictwa Naukowo-Techniczne.

Kordos M. (1994), *Wykłady z historii matematyki,* Warszawa, WSiP.

Marciszewski W. (1998), *Sztuczna inteligencja*, Kraków, Społeczny Instytut Wydawniczy „Znak".

Marciszewski W., Stacewicz P. (2011), *Umysł — Komputer — Świat. O zagadce umysłu z informatycznego punktu widzenia,* Warszawa*,* Akademicka Oficyna Wydawnicza EXIT.

Mioduszewski J. (1996), *Ciągłość*. *Szkice z historii matematyki*, Warszawa, WSiP.

Michalewicz Z. (1992), *Genetic Alghorithms + Data Structures = Evolution Programs*, Berlin, Springer Verlag.

Russel S., Norvig P. (1994), *Artificial Inteligence: A Modern Approach*, Englewood Cliffs, Prentice-Hall.

Stacewicz P. (2010), *Umysł a modele maszyn uczących się. Współczesne badania informatyczne w oczach filozofa*, Warszawa, Akademicka Oficyna Wydawnicza EXIT.

Turing A.M. (1936), *On Computabe Numbers, with an Application to the Entscheidungsproblem*, „Proc. Lond. Math. Soc", (2) 42, s. 230-265; *A Correction*, ibid., 43, 1937, s. 544-546.

Turing A.M. (1995), *Maszyny liczące a inteligencja*, [w:] *Filozofia umysłu*, red. B. Chwedeńczuk, Warszawa 1995, Wydawnictwo Spacja, s. 271-300.

Wirth N. (1989), *Algorytmy + struktury danych = programy*, Warszawa, Wydawnictwa Naukowo-Techniczne.