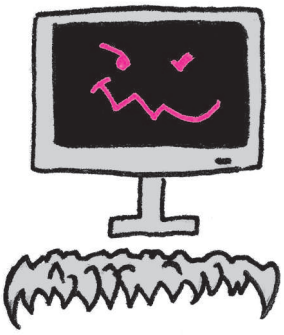

Oblicza informatyki

Władysław M. TURSKI[*]

Informatyka jest nauką o przetwarzaniu informacji, zwłaszcza przy użyciu automatycznych środków pomocniczych.

Słowo „informatyka” wprowadził do polszczyzny nieżyjący już pionier tej dziedziny, profesor Romuald Marczyński, swoim referatem w Zakopanem w grudniu 1968 r. Przedtem używano mniej czy bardziej udolnych kalk anglosaskich: computer science i computing science. Poza ogólnym defektem, polegającym na tym, że dyscypliny mające w swej nazwie „naukę” (science) rzadko są rzeczywiście naukowe, nazwy te były wadliwe dlatego, iż nadmiernie podkreślały rolę narzędzia (computer) lub jednego tylko z wielu sposobów przetwarzania informacji (liczenie – computing). Przyjęta z francuskiego „informatyka” nie ma tych wad i całkiem nieźle odzwierciedla charakter i przedmiot dyscypliny, opisany pierwszym zdaniem niniejszego tekstu.

[*]Instytut Informatyki, Uniwersytet Warszawski

Już starożytni (światową prasę obiegła niedawno fotografia pokładowego komputera z brązu, wydobytego z wraku starogreckiej galery) przetwarzali informacje, posługując się sztucznymi przyrządami. Naturalnym przyrządem, a raczej narządem przetwarzania informacji jest mózg człowieka, żaby czy mrówki, ale naturalne przetwarzanie informacji zachodzi też na poziomie pojedynczych komórek i to wcale niekoniecznie zwierzęcych.

Nie wiemy, niestety, *jak* odbywa się przetwarzanie informacji w istotach żywych i wcale nie jest pewne, że kiedykolwiek się tego dowiemy. Sensacje prasowe o tym, że uczeni kolejny raz „odkryli tajemnicę myślenia”, jeśli nie całkiem fałszywe, dotyczą zjawisk fizjologicznych towarzyszących procesowi myślenia (np. gdy badany myśli o jabłku, zwiększa się elektryczna aktywność neuronów w takiej to a takiej części kory mózgowej). Do zrozumienia istoty myślenia droga stąd wcale nie krótsza, niż od stwierdzenia, że mózg wydziela myśli tak, jak wątroba – żółć.

Jedyne znane nam od A do Z procesy przetwarzania informacji to te, które świadomie wbudowaliśmy do skonstruowanych przez nas urządzeń, i te, które sami czysto mechanicznie wykonujemy zgodnie z ustalonymi procedurami. Doskonale wiemy (a przynajmniej niektórzy z nas wiedzą), jak komputer przetwarza mnożnik i mnożną w iloczyn, ale nikt nie wie, jak odbywa się to w cudzym umyśle. Ba, wielu z nas inaczej znajduje wynik mnożenia $7 \cdot 8$ niż mnożenia $10 \cdot 5$, a komputer robi to zawsze (w pewnych granicach) tak samo.

Niektórzy, bardzo łatwowierni ludzie uważają, że skoro i komputer, i człowiek udzielają tej samej odpowiedzi na pytanie o iloczyn $7 \cdot 8$, i skoro wiadomo, że człowiek myśli, to komputer też myśli, jest obdarzony tzw. sztuczną inteligencją. Inni wyciągają taki wniosek z faktu, że komputery potrafią wygrywać z arcymistrzami szachów. I w tym przypadku autorzy programu doskonale wiedzą, krok po kroku, jak „gra” komputer, nikt zaś nie ma najmniejszego pojęcia o tym, jak gra arcymistrz, a nawet taki antytalent szachowy, jak ja. Między tymi dwoma sposobami przetwarzania informacji: myśleniem i wykonywaniem programu zachodzi fundamentalna różnica i nic, jak dotąd, jej nie zmniejszyło ani o jotę, choć co do wyników są one czasem zupełnie zgodne, a komputerowe bywają niekiedy o niebo doskonalsze.

Przez długi czas głównym przedmiotem zainteresowania informatyki i informatyków były obliczenia numeryczne, czyli rachunki. Zastosowanie komputerów umożliwiło wykonywanie rachunków, jakie, choć koncepcyjnie proste, były dla ludzi praktycznie niewykonalne. Tytułem przykładu, wyznaczanie orbit poszczególnych komet, co jeszcze w czasach mojej wczesnej młodości było zajęciem niebywale czasochłonnym i stanowiło osiągnięcie uprawniające do obejmowania katedr uniwersyteckich, niemalże z dnia na dzień stało się sprawą rutynową. Bez bardzo szybkich obliczeń nie byłoby astronautyki, nowoczesnej broni i Złotych Tarasów. Możliwość szybkiego rachowania zrodziła też zupełnie nowe metody badań naukowych i nowe praktyki inżynierskie i medyczne. Bez niej nie byłoby metody elementu skończonego, powszechnie dziś stosowanej w projektowaniu inżynierskim, nie miałaby sensu szybka transformata Fouriera (FFT), bez której nie byłoby tomografii komputerowej i wielu innych ważnych, codziennych zastosowań w medycynie (i nie tylko). Bez szybkiego liczenia nie zafascynowałaby nas uroda fraktali (bo byśmy ich w ogóle nie zobaczyli!), nie byłoby impulsu dla badań chaosu, a tym samym – wielu nowoczesnych technologii, nie mówiąc już o doskonałości barw, cieni i faktury grafik komputerowych, ani awatarów w grach i zabawach. Nie byłoby także MP3 i wszystkiego, co za tym idzie.

Kiedy głównym przedmiotem zainteresowania informatyki było liczenie, najważniejszym problemem, obok poprawności rachunku, było pytanie, czy zadane obliczenia kiedykolwiek dobiegną pomyślnego końca? To wcale nie jest pytanie banalne: gdy liczy się coś zupełnie nowatorskiego i wiele godzin czeka na wynik, to bardzo chce się wiedzieć, czy to coś jeszcze nie wyszło, czy też program „zapętlił się na amen”, wykonuje te same instrukcje w kółko, nie posuwając się „do przodu”. I tu spotkaliśmy się – bodajże po raz pierwszy w intelektualnych dziejach ludzkości! – z prostym, lecz nierozstrzygalnym problemem

praktycznym. Owszem, znany już był wynik Gödla o nierozstrzygalności arytmetyki i różnych logik, ale po pierwsze, interesowało się nim bardzo niewiele osób, a po drugie – problemy te nosiły nieco metafizyczny charakter, a w metafizyce wiadomo, że często nie wiadomo, jak jest „naprawdę". Teraz dowiedzieliśmy się (z dowodem na mur!), że nie da się skonstruować takiego programu, który mógłby posłużyć do badania dowolnego innego programu na okoliczność: czy jego wykonanie skończy się, czy nie.

Nierozstrzygalność problemu stopu (taką nazwę nosi opisane powyżej zadanie) wcale nie oznacza, że dla konkretnego programu nie można stwierdzić, czy jego wykonanie kończy się, czy nie, czy dzieje się tak zawsze, czy tylko w pewnych warunkach. Badania empiryczne (polegające na „puszczaniu" programu na komputerze) nie są tu przydatne, zwłaszcza wtedy, gdy program może nie mieć stopu. Powstało więc zapotrzebowanie na umiejętność wnioskowania o własnościach programu na podstawie analizy jego tekstu, a nie zachowania się w trakcie wykonywania. W ten sposób zrodziła się nowa dyscyplina informatyki: logika programów. Jej naturalnym rozwinięciem stało się programowanie bezpośrednio w logice.

Jak widać, komputery można programować na różne sposoby: żeby udawały rachmistrza i żeby naśladowały czysto logiczne wnioskowanie. Do sprawnego opisu intencji programisty powstały dziesiątki (jeśli nie setki) języków programowania. Niegdyś toczono „święte wojny" o to, który język programowania jest najlepszy. Dziś wiemy już, że pytanie to nie ma większego sensu, gdyż użyteczność języka zależy od szerokiego kontekstu, od celu, środowiska i predylekcji tego, kto ma programować.

Rozmaitość języków programowania świadczy o nabytej łatwości tworzenia translatorów, tj. programów tłumaczących z języka programowania na język sterowania komputerem. Informatyka dochodziła do tego dość długo; trzeba było zbudować nowe teorie, opracować wiele sprytnych metod. Dziś w zasadzie dzieło to zostało zakończone i opisane w solidnych podręcznikach, nowej wiedzy w tym obszarze wiele już nie przybywa. Z jednym, szalenie ważnym wyjątkiem.

Jedną z podstaw wiedzy o językach programowania jest teoria języków formalnych, tj. ciągów powstających z zadanego alfabetu przez stosowanie określonego repertuaru reguł. Teoria ta rozwinęła się bardzo pięknie, jednakże od pewnego czasu zaczęła tracić znaczenie: jej interesujące nowe odkrycia nie miały już żadnego związku z praktyką, co w informatyce jest oznaką dekadencji. Okazało się jednak, że przed tą teorią najfantastyczniejsze wyzwania postawiła . . . przyroda. Spirala DNA, owa alfa i omega wiedzy o życiu, jest przecież napisem zbudowanym z czterech symboli. Rozszyfrowanie reguł, wedle których ten napis jest zbudowany, wyszukiwanie identycznych fragmentów w dwu i więcej różnych spiralach, odtwarzanie ewolucji itp. – wszystko to są zadania o kolosalnym znaczeniu praktycznym, a biegłość w stosowaniu metod teorii języków formalnych – silnym atutem w ich rozwiązywaniu. Nowiutka gałąź informatyki – bioinformatyka – pełnymi garściami czerpie z dorobku teorii języków formalnych i stymuluje jej dalszy rozwój.

Komputery różnią się między sobą pod wieloma względami. Aby móc badać ogólne własności programów, należało wymyślić swego rodzaju wzorzec, do którego dałoby się sprowadzić każdy realny komputer. Takim wzorcem jest abstrakcyjna maszyna Turinga (wymyślona przed zbudowaniem pierwszych komputerów!); każde realne obliczenie można sprowadzić do obliczenia na maszynie Turinga. Z takim instrumentarium badawczym udało się podzielić problemy rachunkowe na klasy, charakteryzujące się swoistą złożonością obliczeniową. Dla przykładu, w jednej klasie są wszystkie problemy, których rozwiązania wymagają ilości rachowania rosnącej jak wielomian rozmiaru zadania, w innej – problemy, których apetyt na rachunki rośnie szybciej niż jakikolwiek wielomian. Przynależność problemu do danej klasy nie zależy od metody jego rozwiązania; jeśli więc dla problemu z klasy wielomianowej mamy program, który w miarę zwiększania rozmiaru zadania pochłania „nadwielomianowo" wiele mocy obliczeniowej, są podstawy wątpić w jego jakość. Badanie złożoności obliczeniowej różnych problemów stanowi wciąż żywą dyscyplinę informatyki. Do dziś nie wiemy na przykład, czy dwie najważniejsze bodaj klasy obliczeń są, czy nie są tożsame; na autora dobrze uzasadnionej odpowiedzi od lat czeka sowita nagroda!

Z biegiem czasu, oprócz klasycznych obliczeń, takich z początkiem (danymi) i końcem (wynikiem), informatyków zaczęły interesować procesy innego typu. Weźmy dla przykładu system operacyjny – taki jak Windows czy Linux. Z reguły wcale nie jesteśmy zainteresowani stopem tego programu, przeciwnie, jego spontaniczne zatrzymanie (zawieszenie) się jest źródłem wielu zmartwień. Celem tego typu procesu nie jest już uzyskanie wyniku, lecz zachowanie określonych relacji miedzy wskazanymi parametrami, co wymaga właściwego reagowania na ich zmiany, zachodzące spontanicznie (np. gdy użytkownik naciśnie klawisz), albo pod wpływem realizacji innych programów. Jądrem problematyki wynikającej z tego rodzaju okoliczności jest utrzymanie celowej (pożądanej) współpracy wielu biegnących równocześnie autonomicznych procesów. Że nie są to problemy łatwe, poucza smutny los ośliny, co to pośród jadła z głodu padła i wzajemne blokowanie się dwu gentlemanów, upierających się, by przepuścić drugiego przez wąskie przejście.

Przykładem systemu obejmującego olbrzymią liczbę niezależnych, a współdziałających procesów jest oczywiście Internet, sieć łącząca setki milionów autonomicznych komputerów, współpracujących ze sobą przez wysyłanie i odbieranie adresowanych komunikatów, składających się z pakietów informacji, zbudowanych zgodnie z powszechnie stosowanym

protokołem. Komunikatem może być praktycznie wszystko: list miłosny, fragment zakodowanego filmu, wirus komputerowy, sygnał uruchamiający ogrzewanie w moim domu itp. Nie wszystkie pakiety składające się na komunikat idą tą samą drogą od nadawcy do odbiorcy; węzły transmisji wybierają dla każdego z osobna drogę najtańszą. W rezultacie, pakiety mogą docierać nie po kolei, ostateczny montaż odbywa się w komputerze odbiorcy. To, że korzystając z Internetu, mamy wrażenie pełnej ciągłości i płynności dźwięku i obrazu, świadczy o mocy sprzętowych środków informatyki (hardware) i skuteczności oprogramowania (software) tworzących światową sieć komputerową (WWW).

Protokoły internetu są jednakowe dla wszystkich. Stosują je uczniowie i hobbyści, uczeni i maklerzy, terroryści i policjanci. Umożliwia to globalne funkcjonowanie Internetu, lecz także, niestety, ułatwia stale rosnącą przestępczość w sieci. Paradoksalnie, gdyby sieć nie rozwinęła się tak szybko, a raczej gdyby jej rozwój nie nastąpił tak wcześnie, sytuacja mogłaby być inna.

Od drugiej wojny światowej kryptologia (wiedza o szyfrowaniu) zmieniła swój charakter. Od czasu, gdy M. Rejewski z kolegami czysto matematycznym wnioskowaniem (opartym na teorii grup) złamali szyfr Enigmy i zbudowali mechaniczne szablony (dla niepoznaki i żartu nazwane bombami) do masowego dekodowania niemieckich depesz wojskowych, amerykańscy kryptolodzy też przez czysto matematyczną analizę złamali japońskie szyfry i wreszcie A. Turing zbudował (w 1943/44) elektroniczny komputer (Colossus, tajny do lat 80. ub. w.) do deszyfrowania niemieckich depesz kodowanych ulepszonymi wersjami Enigmy i zupełnie nowymi Lorentzami – stało się oczywiste, że kryptologia wychodzi z epoki płaszcza i szpady i staje się gałęzią matematyki i informatyki. Wzbudziła ożywione zainteresowanie analityczną teorią liczb, którą mało kto się zajmował, przywróciła blask teorii funkcji eliptycznych, od stu bez mała lat uważanej za zamkniętą księgę, zapoczątkowała prace nad konstrukcją stosunkowo tanich urządzeń do szybkiego operowania na bardzo wielocyfrowych liczbach całkowitych. Dwa wspaniałe odkrycia kryptologii: asymetryczna kryptografia z kluczem publicznym (pozwalająca ujawniać klucz do kodowania, zachowując w tajemnicy klucz do rozkodowania) i podpis elektroniczny (dający gwarancję autentyczności dokumentu i jego autorstwa) zmieniłyby oblicze Internetu, szalenie utrudniając życie oszustom i figlarzom, gdyby tylko były powszechnie dostępne wtedy, gdy tworzyły się protokoły sieciowe. Dodanie ich do sieci teraz wymaga sporych modyfikacji i jest zbyt kosztowne na to, żeby zyskać popularność.

Rozwiązanie zadania F 699.
W czasach Perelmana sądzono, iż piorun realizuje się średnio przy różnicy potencjału 1000 MV, dziś encyklopedie podają tylko 100 MV. Co do natężenia prądu w piorunie, dane są jeszcze bardziej płynne; podaje się, że osiąga średnio 20 kA. Daje to łącznie moc 2 miliony MW, czyli 2 miliardy kW, z czego należy do rachunku wpisać połowę, gdyż potencjał podczas wyładowania spada do zera. Wyładowanie trwa średnio tylko 10^{-4} s, co stanowi $28 \cdot 10^{-9}$ godziny. Łącznie Zeus musi zatem zapłacić za około 28 kWh, czyli 7 euro. Nic więc dziwnego, że – nawet jeśli ma nie najwyższą emeryturę – pozwala sobie czasami na długotrwałe burze.

Rozwiązanie zadania M 1180.
Zauważmy, że

$$(ac+bd)^2+1 = \\ = (ac+bd)^2+(ad-bc)^2 = \\ = (a^2+b^2)(c^2+d^2)\,.$$

Jeśli więc $p>0$ jest wspólnym dzielnikiem liczb $ac+bd$ i a^2+b^2, to p jest dzielnikiem liczby 1, czyli $p=1$. To oznacza, że liczby $ac+bd$ i a^2+b^2 są względnie pierwsze.

Najprostszy algorytm sortowania

Rozważmy problem sortowania n-elementowej tablicy `a[1..n]`. Jaki algorytm pozwoli nam zrobić to najprościej? Nie chodzi tu wcale o szybkość, ale o to, by dało się go zaprogramować w kilku linijkach, bez żadnego ryzyka popełnienia błędu, nawet jeśli nie jesteśmy akurat w najlepszej dyspozycji umysłowej (zaraz, zaraz, miałem porównać `a[i]` z `a[i+1]` czy z `a[i-1]`? czy wewnętrzna pętla miała się obracać, dopóki `i<j`, czy `i<=j`? oj...).

No więc, który algorytm jest najprostszy? Chyba żaden z działających w czasie $O(n \log n)$ się nie nadaje. To może sortowanie przez wstawianie albo bąbelkowe? Nic z tego. Najprościej jest tak:

```
sort(a[1..n]) {
    for i:=1 to n do
    for j:=1 to n do
      if a[i] < a[j] then zamień(a[i], a[j]);
}
```

Szybki test: czy ten algorytm sortuje tablicę `a` rosnąco czy malejąco? Nie jest to jasne na pierwszy rzut oka, podobnie jak to, że tablica w ogóle zostanie jakkolwiek posortowana!

Po chwili zastanowienia stwierdzamy, że powyższy algorytm to jedno z wcieleń sortowania przez wstawianie (Insertion Sort). Na początku i-tego obrotu zewnętrznej pętli fragment `a[1..i-1]` jest posortowany (rosnąco), a wewnętrzna pętla wstawia w odpowiednie miejsce element `a[i]`, używając i-tej pozycji w tablicy jako „bufora” na przesuwane elementy początkowego fragmentu. Czytelnik łatwo sprawdzi, że wewnętrzna pętla mogłaby zakończyć działanie dla `j=i`, ale jej przedłużenie aż do `j=n` nic nie popsuje.

Nie jest to specjalnie efektywne, ale za to jakie eleganckie! Prościej już się chyba nie da...

Michał ADAMASZEK