Bajka o złożoności obliczeniowej i sprytnej Agatce

Za siedmioma górami, za siedmioma rzekami – gdzieś pod Warszawą – znajduje się niewielka miejscowość. W tej miejscowości stoi mały domek. A tak się składa, że w domku tym mieszkają Bartek i Agatka wraz z rodzicami.

Bartek jest starszy od Agatki. Chodzi do prestiżowego liceum w stolicy i startuje w różnych konkursach programistycznych. Rodzice są bardzo dumni z Bartka i na ostatnie urodziny kupili mu wyjątkowo drogi komputer do nauki. Komputer ten ma procesor, pamięć RAM i wszystkie inne bajery, jakie tylko można sobie wyobrazić.

Agatka jest bardzo zapatrzona w brata. Chce być taka jak on. Od Bartka dostała jego stary komputer. Wprawdzie klawiatura jest brudna, komputer jest bardzo przestarzały i często się zawiesza, jednak nie powstrzymuje to Agatki przed nauką programowania.

Ostatnio Agatka znalazła w bibliotece książkę dotyczącą podstaw algorytmiki. Pierwszy rozdział książki był poświęcony złożoności obliczeniowej. Autorzy książki tłumaczyli, że są dwie metody sprawdzania, który algorytm działa szybciej. Pierwsza z nich to metoda empiryczna. Polega ona na napisaniu dwóch programów, uruchomieniu ich na danych testowych i zmierzeniu czasu każdego z nich. Metoda ta ma wiele wad. Różne wyniki można otrzymać w zależności od tego, na jakim komputerze uruchomimy program, jaka będzie architektura procesora, jaka będzie struktura pamięci komputera, jakich kompilatorów użyjemy, w jakich językach napiszemy programy, jaki będzie rozmiar danych testowych, jakie dane testowe użyjemy czy jakie liczby wygeneruje generator liczb losowych. Mówiąc prościej – wynik eksperymentu może zależeć od wielu czynników. Przede wszystkim jednak wadą tej metody jest to, że najpierw należy oba programy napisać na komputerze – co w przypadku skomplikowanych algorytmów może okazać się czasochłonne.

Drugą metodę autorzy tłumaczą na przykładzie poniższego programu:

`wczytaj a`	m_1	1
`wczytaj b`	m_1	1
`c := 0`	m_2	1
`dopóki a > 0`	m_3	a
`  b' := b`	m_4	a
`  dopóki b' > 0`	m_3	$a \cdot b$
`    c := c + 1`	m_5	$a \cdot b$
`    b' := b' - 1`	m_6	$a \cdot b$
`  a := a - 1`	m_6	a
`wypisz c`	m_7	1

Jeśli chcielibyśmy ustalić, ile czasu zajmie komputerowi obliczenie powyższego programu, musielibyśmy wiedzieć, ile milisekund zajmuje mu wykonanie każdej z instrukcji. Ponieważ to zależy od komputera

Milisekunda to jedna tysięczna część sekundy.

(a jak napisali autorzy książki: „my chcemy zajmować się algorytmami, a nie maszynami liczącymi"), ustalimy sobie pewne stałe. Załóżymy, że instrukcja `wczytaj a` wykona się na komputerze w m_1 milisekund, instrukcja `c := 0` wykona się w m_2 milisekundy, i tak dalej. Następnie chcemy policzyć, ile razy (w najgorszym przypadku) wykonywana będzie dana instrukcja przez komputer. Dla przykładu: instrukcja `wczytaj a` wykona się 1 raz, natomiast instrukcja `c := c + 1` wykona się $a \cdot b$ razy. Można obliczyć, że program będzie wykonywać się przez następującą liczbę milisekund:

$$m_1 + m_1 + m_2 + a \cdot m_3 + a \cdot m_4 + a \cdot b \cdot m_3 + a \cdot b \cdot m_5 + a \cdot b \cdot m_6 + a \cdot m_6 + m_7,$$

co można zapisać krócej jako:

$$a \cdot b \cdot (m_3 + m_5 + m_6) + a \cdot (m_3 + m_4 + m_6) + 2 \cdot m_1 + m_2 + m_7.$$

Teraz będziemy się zastanawiać, co się będzie działo, gdy wartości a oraz b będą bardzo duże. Można zauważyć wtedy, że pierwszy i drugi składnik będą tak dużo większe od trzeciego, że trzeci będzie w porównaniu z nimi pomijalnie mały. Pomińmy go zatem:

$$a \cdot b \cdot (m_3 + m_5 + m_6) + a \cdot (m_3 + m_4 + m_6).$$

Teraz spróbujemy sobie wyobrazić, co się stanie, gdy wartości a oraz b będą naprawdę bardzo, bardzo duże. Wtedy pierwszy składnik sumy będzie na tyle duży, że wartość drugiego składnika stanie się w porównaniu z nim pomijalnie mała. Zatem i ją pomińmy:

$$a \cdot b \cdot (m_3 + m_5 + m_6).$$

Ponieważ, jak wielokrotnie autorzy książki już wspominali, chcemy zajmować się algorytmami, a nie komputerami – pomińmy dodatkowo współczynnik w nawiasie:

$$a \cdot b.$$

Otrzymaliśmy coś, co informatycy nazywają asymptotyczną złożonością obliczeniową algorytmu. Mówi się czasem, że algorytm działa w czasie $O(a \cdot b)$, albo że algorytm ma złożoność $O(a \cdot b)$. Metoda ta ma dwie podstawowe zalety. Po pierwsze – bardzo łatwo ją zastosować. Tak naprawdę nie potrzebujemy nawet powtarzać wszystkich tych kroków, które poczyniliśmy. Wystarczy, że spojrzymy na algorytm i zastanowimy się, która instrukcja będzie wykonywana najczęściej przez program. W naszym przykładzie jest to linijka `c := c + 1` i faktycznie jest ona wykonywana dokładnie $a \cdot b$ razy. Po drugie, łatwo na podstawie złożoności określić, który algorytm będzie działał szybciej. I to jeszcze przed napisaniem go na komputerze! Dla przykładu: algorytm działający w czasie $O(a \cdot b)$ działa wolniej od algorytmu $O(a + b)$, a ten z kolei od algorytmu ze złożonością $O(a)$. Oczywiście, metoda ta ma również wady. Po pierwsze, mówi ona, co się dzieje dla dużych danych. O tym, który algorytm działa szybciej dla małych danych, nie mówi nic. Po drugie, jeśli dwa algorytmy mają taką samą złożoność – nie dowiemy się, który działa krócej.

Dodatkowo pomija się stałe lub niektóre elementy sumy. Na przykład, jeśli najczęściej wykonywaną instrukcję komputer wykonywałby $2 \cdot a \cdot b + a$ razy, to złożoność wciąż wynosiłaby $a \cdot b$.

Uzbrojona w nową wiedzę Agatka postanowiła wyzwać swojego brata na pojedynek. Założyła się z bratem, że jej program sortujący zadziała na jej wolnym komputerze szybciej niż program sortujący Bartka na jego superkomputerze. Bartek bez zastanowienia przyjął zakład. Napisanie programu zajęło mu 5 minut. Agatce natomiast zajęło to cały dzień. Bartek nie wiedział jednak, że Agatka wie, że Bartek zna tylko jeden algorytm sortowania. W książce przeczytała też, że złożoność tego algorytmu to $O(n^2)$. W rozdziale dalej z kolei był podany algorytm sortowania o złożoności $O(n \log_2 n)$.

Logarytm to operacja odwrotna do potęgowania: $\log_2 a = b \iff a = 2^b$.

Kto wyjdzie z tego pojedynku zwycięsko? Bartek jest świetnym programistą. Stała ukryta w złożoności programu Bartka wynosi 1/10. Ponadto Bartek tak zaprogramował swój algorytm, że jest on w stanie pracować równolegle na wszystkich czterech rdzeniach jego komputera bez żadnych

dodatkowych narzutów czasowych. Każdy z rdzeni jego komputera taktuje z częstotliwością 2,5 GHz. Agatka nie jest jeszcze taką dobrą programistką jak jej starszy brat. Stała ukryta w złożoności programu Agatki wynosi 20. Komputer Agatki to bardzo stary Commodore 64 z procesorem taktującym z częstotliwością 1 MHz. Wydawać by się mogło, że Agatka nie ma żadnych szans. Jednak sprytna siostra zażądała, żeby sortowali oboje wszystkich ludzi na świecie. W sumie około 8 000 000 000 nazwisk.

Komputery będą działały długo i trochę wody w Wiśle upłynie, zanim rodzeństwo dowie się, kto wygrał zakład. My to obliczymy już teraz. Przyjmując dodatkowe założenia, możemy obliczyć, że program Agaty będzie się liczył około $3{,}6 \cdot 10^6$ sekund, natomiast program Bartka $6{,}4 \cdot 10^8$ sekund. Mówiąc bardziej obrazowo: program Agatki skończy się liczyć w półtora miesiąca, natomiast program Bartka będzie się liczył ponad 20 lat.

$(1/10) \cdot (8 \cdot 10^9)^2/10$ GHz $= 64 \cdot 10^{17}/10^{10} = 6{,}4 \cdot 10^8$

Krzysztof PIECUCH

Resztki

– *Skończyłam!* – krzyknęła triumfalnie Agatka do swojego brata, Bartka. Dziewczynka regularnie domaga się od starszego chłopca rozmaitych ciekawostek matematycznych, których ten dowiaduje się w liceum. Tym razem Bartek, aby uzyskać chwilę spokoju, przykazał jej (twierdząc, że jest w tym jakiś głębszy sens) umieścić w tabelce 21×10 liczby od 1 do 210 w taki sposób, aby numery wiersza i kolumny, w jakich znajdzie się dana liczba, odpowiadały jej resztom z dzielenia odpowiednio przez 21 i 10. – *To było dość żmudne i jakoś nie wydaje mi się, by kryło się tu coś ciekawego... na pewno nie chciałeś się mnie po prostu pozbyć na chwilę?*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**0**
1	1	22	43	64	85	106	127	148	169	190
2	191	2	23	44	65	86	107	128	149	170
3	171	192	3	24	45	66	87	108	129	150
4	151	172	193	4	25	46	67	88	109	130
5	131	152	173	194	5	26	47	68	89	110
6	111	132	153	174	195	6	27	48	69	90
7	91	112	133	154	175	196	7	28	49	70
8	71	92	113	134	155	176	197	8	29	50
9	51	72	93	114	135	156	177	198	9	30
10	31	52	73	94	115	136	157	178	199	10
11	11	32	53	74	95	116	137	158	179	200
12	201	12	33	54	75	96	117	138	159	180
13	181	202	13	34	55	76	97	118	139	160
14	161	182	203	14	35	56	77	98	119	140
15	141	162	183	204	15	36	57	78	99	120
16	121	142	163	184	205	16	37	58	79	100
17	101	122	143	164	185	206	17	38	59	80
18	81	102	123	144	165	186	207	18	39	60
19	61	82	103	124	145	166	187	208	19	40
20	41	62	83	104	125	146	167	188	209	20
0	21	42	63	84	105	126	147	168	189	210

– *Ależ skąd!* – odpowiedział brat z udawanym oburzeniem. – *Zauważ najpierw, że żadne dwie liczby nie zostały wpisane w tę samą komórkę. Gdyby bowiem tak się stało, to te dwie liczby dawałyby tę samą resztę z dzielenia przez* 10 *i* 21*. W tej sytuacji ich różnica byłaby podzielna przez* 10 *i* 21*, a zatem przez* 210 *(byłaby więc zerem), gdyż...* – i tu Bartek teatralnie zawiesił głos.

– *... gdyż są to liczby względnie pierwsze!* – dokończyła prędko Agatka, ponieważ niedawno omawiali ten temat na kółku matematycznym. Po chwili dodała: – *A skoro zarówno liczb, jak i komórek jest* 210*, więc w każdej komórce wyląduje jakaś liczba!*

– *Doskonale.* – pochwalił siostrę Bartek. – *Udowodniłaś właśnie Chińskie Twierdzenie o Resztach: każdy układ reszt z dzielenia przez parami względnie pierwsze liczby jest możliwy do zrealizowania.*
A skoro jesteśmy przy liczbach względnie pierwszych, zwróć uwagę na kolejną rzecz. Otóż jeśli wybierzesz dowolną liczbę względnie pierwszą z 210*, to jej wierszowa współrzędna jest względnie pierwsza z* 21*, a kolumnowa z* 10 *i odwrotnie: każda taka para współrzędnych określa liczbę względnie pierwszą z* 210 *(dowód nie jest trudny, spróbuj sama!).* – mówiąc to, Bartek zamalował na kolorowo wszystkie liczby, które nie były względnie pierwsze z 210. – *W tej sytuacji, jeśli przez* $\varphi(n)$ *oznaczymy liczbę liczb mniejszych od* n *i względnie pierwszych z* n*, to musi zachodzić* $\varphi(210) = \varphi(10) \cdot \varphi(21)$*. Podobna zależność zachodzi z tych samych względów dla iloczynu dowolnych dwóch liczb względnie pierwszych.*

– *Wspaniale!* – wykrzyknęła Agatka. – *W tej sytuacji* $\varphi(210)$ *wynosi* $1 \cdot 4 \cdot 2 \cdot 6$*, czyli* 48*. Nie mogłeś mi tego wszystkiego powiedzieć bez tej upiornej tabelki...?*

Łukasz RAJKOWSKI