O złożoności algorytmów

Mgr Teresa PRZYTYCKA

Rozważając algorytm, czyli przepis automatycznego postępowania prowadzący do rozwiązania określonego zadania, chcielibyśmy zazwyczaj wiedzieć, jak szybko uzyskamy rozwiązanie stosując właśnie ten algorytm. Pozwoli nam to wybrać spośród wielu algorytmów możliwie najlepszy i, co nie mniej ważne, oszacować jego realizowalność na dostępnej nam maszynie. Załóżmy np., że maszyna wykonuje 10^5 kroków na sekundę, a liczba kroków naszego algorytmu jest funkcją danej n i wyraża się przez n^n. Wówczas dla $n = 5$ maszyna będzie pracowała ułamek sekundy, dla $n = 10$ około doby, dla $n = 11$ ponad 33 dni, podczas gdy dla $n = 14$ powyżej 3000 lat! Jeśli więc liczba kroków algorytmu rośnie tak szybko wraz ze wzrostem n, to już dla całkiem małych wartości n algorytm praktycznie nie jest realizowalny.

Oceny czasochłonności algorytmu dokonujemy na podstawie jego funkcji złożoności czasowej. Jest to funkcja przyporządkowująca rozmiarowi danych liczbę kroków algorytmu.

Jeżeli dopuszczamy możliwość wystąpienia różnych danych o tym samym rozmiarze (porównaj uwagę na marginesie), to przyporządkowując rozmiarowi danych maksymalną liczbę kroków wykonywanych dla danych tego rozmiaru otrzymamy funkcję pesymistycznej złożoności czasowej (T_{pes}).

W przypadku, gdy rozmiarowi przyporządkowujemy średnią liczbę kroków po zbiorze danych tego rozmiaru (dokładniej — wartość oczekiwaną, porównaj margines), otrzymujemy funkcję oczekiwanej złożoności czasowej (T_{ave}).

W praktyce nie interesuje nas dokładna postać funkcji złożoności czasowej, ale jej rząd. Chcemy bowiem wiedzieć, jak szybko rośnie liczba kroków wraz ze wzrostem rozmiaru danych. Pytamy więc, czy rośnie ona tak szybko jak funkcja n^2, czy jak $n!$, czy też jak $\log n$, a nie czy jest to dokładnie $\frac{1}{2}n^2+7$, czy $n!-8$, czy też $120 \log n$. Zamiast wszystkich kroków algorytmu możemy więc liczyć liczbę operacji dominujących, przy czym za operacje dominujące przyjmujemy takie operacje, że funkcja przyporządkowująca rozmiarowi danych liczbę tych operacji jest rzędu funkcji złożoności czasowej.

Rozważmy dla przykładu różne algorytmy dla zadania sortowania. Zadanie to można sformułować w następujący sposób: Dany jest ciąg n liczb $a_1, a_2, \ldots, a_n$; dokonać takiej permutacji tego ciągu, aby $a_1 \leqslant a_2 \ldots \leqslant a_n$. W zadaniu tym rozmiarem danych będzie długość ciągu, we wszystkich zaś prezentowanych algorytmach operacją dominującą będzie porównanie dwu elementów ciągu. Przy analizie funkcji złożoności będziemy zakładać, że $a_1, \ldots, a_n$ jest permutacją n różnych liczb oraz że dla ustalonego n każda permutacja jest jednakowo prawdopodobna.

Pierwszy z proponowanych algorytmów polega na znalezieniu największego elementu w danym ciągu, potem drugiego co do wielkości itd. Metoda ta nosi nazwę sortowania przez wybór.

Algorytm 1: Sortowanie przez wybór
$i := n$
dopóki $i > 1$ powtarzaj:
  {wybieramy największy element z podciągu $a_1, \ldots, a_i$}
  $max := 1$;
  {max jest indeksem największego z rozważonych dotychczas elementów}
  $k := 2$;
  dopóki $k \leqslant i$ powtarzaj:
    jeśli $a_k > a_{max}$ to $max := k$;
    $k := k+1$;
  {zamieniamy miejscami a_i z a_{max}}
  $x := a_{max}$; $a_{max} := a_i$; $a_i := x$;
  $i := i-1$;
  {teraz mamy następującą sytuację $a_1, \ldots, a_{i-1}, a_i \leqslant a_{i+1} \leqslant \ldots \leqslant a_n$}

Obok pokazujemy, jak działa ten algorytm dla ciągu 1, 5, 4, 2, 3.

$i=n=5$	1 5 4 2 3	max przyjmuje kolejno wartości 1,2
$i=4$	1 3 4 2 5	max przyjmuje kolejno wartości 1,2,3
$i=3$	1 3 2 4 5	max przyjmuje kolejno wartości 1,2
$i=2$	1 2 3 4 5	max przyjmuje kolejno wartości 1,2
$i=1$	koniec działania	

Poszukajmy funkcji złożoności tego algorytmu. Zauważmy, że dla ustalonej długości ciągu liczba porównań będzie zawsze taka sama. Zatem

$$T_{pes} = T_{ave} = \sum_{i=2}^{n}\sum_{k=2}^{i} 1 = \sum_{i=2}^{n}(i-1) = \sum_{i=1}^{n-1} i = \frac{1+n-1}{2}\cdot(n-1) = \frac{n^2}{2} - \frac{n}{2}.$$

Zatem funkcja złożoności czasowej (zarówno średniej, jak i pesymistycznej) jest rzędu n^2.

Aby poprawić średnią złożoność czasową, postarajmy się znaleźć taki algorytm, który w przypadku, gdy dany ciąg jest już częściowo posortowany, będzie wykonywał mniej kroków. Zastosujmy metodę podobną jak przy układaniu kart do gry. Każdą kolejną kartę wkładamy między karty już wcześniej ułożone.

Dla każdego problemu algorytmicznego możemy określić funkcję ω, która danym dla tego problemu przyporządkowuje ich wymiar. Niech D będzie zbiorem danych, W — zbiorem wymiarów. Zatem $\omega: D \to W$. Np. dla zadania znajdowania minimalnego elementu w ciągu n elementów D będzie zbiorem ciągów skończonych, a ω będzie przyporządkowywać ciągowi jego długość ($W = N$).

Niech $f: D \to N$ i $f(d)$ będzie liczbą kroków dla danych d, zaś p_{dn} będzie prawdopodobieństwem wystąpienia danej d w zbiorze danych o rozmiarze n ($n \in W$). Wówczas

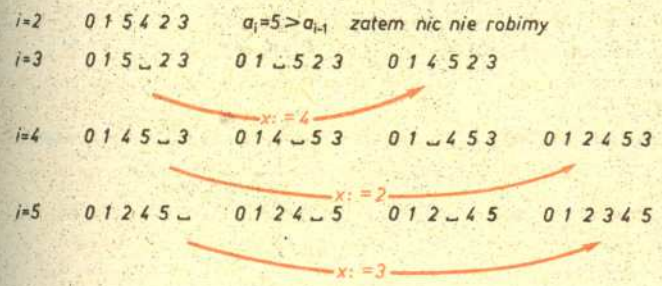
$$T_{pes}(n) = \sup\{f(d) : \omega(d) = n\},$$

$$T_{ave}(n) = \sum_{\omega(d)=n} f(d)p_{dn}.$$

Załóżmy, że ciąg $a_1, a_2, \ldots, a_{i-1}$ jest posortowany i chcemy wstawić i-ty element we właściwe miejsce. W tym celu przeglądamy ciąg $a_1, \ldots, a_{i-1}$ od końca aż do napotkania pierwszego elementu mniejszego od elementu, który chcemy wstawić. Aby taki element zawsze się znalazł, dokładamy na początku ciągu element $a_0 = 0$. Za tak znaleziony element wstawiamy element i-ty. Otrzymany ciąg $a_1, a_2, \ldots, a_{i-1}, a_i$ jest posortowany i o ile $i < n$ możemy czynność powtórzyć.

Algorytm 2: Sortowanie przez wstawianie

```
i := 2;
dopóki i ⩽ n powtarzaj:
| {a_0 ⩽ a_1 ⩽ ... ⩽ a_{i-1}}
| jeśli [a_{i-1} > a_i] to
| | {a_0 ⩽ a_1 ⩽ ... ⩽ a_{i-1} i a_{i-1} > a_i, wstawiamy i-ty element}
| | x := a_i;
| | j := i-1;
| | dopóki [a_j ⩾ x] powtarzaj:
| | | a_{j+1} := a_j;
| | | {przesuwamy element a_j o jedno miejsce w prawo}
| | | j := j-1;
| | {a_j < x i wszystkie elementy większe od x zostały przesunięte
| | o jeden w prawo}
| | a_{j+1} := x;
| i := i+1;
```



Obok pokazujemy, jak będzie wyglądało takie sortowanie ciągu 1, 5, 4, 2, 3.

Zostały przyjęte następujące reguły zapisu algorytmów:

1. Instrukcję „x otrzymuje wartość y", zwaną w informatyce instrukcją podstawienia, zapisujemy symbolicznie $x := y$.
2. Poszczególne instrukcje oddzielamy średnikami.
3. Klamrą [obejmujemy ciąg instrukcji, które chcemy traktować jako jedną całość. Klamra ta będzie najczęściej obejmować ciąg instrukcji, który chcemy powtórzyć pewną liczbę razy lub też wykonać, jeśli spełniony jest określony warunek.
4. W nawiasach klamrowych { } umieszczamy komentarze wyjaśniające poszczególne fragmenty algorytmu.
5. Operacje dominujące będziemy obwodzić ramką.

Łatwo zauważyć, że najwięcej porównań trzeba będzie wykonać w przypadku, gdy ciąg jest posortowany w odwrotnej kolejności.

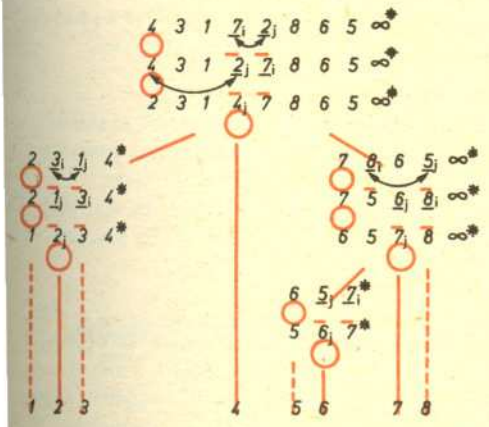
$$T_{pes} = \sum_{i=2}^{n}\left(1+\sum_{j=0}^{i-2}1\right) = \sum_{i=2}^{n}(1+i-1) = \frac{2+n}{2}(n-1) = \frac{1}{2}n^2+\frac{1}{2}n-1.$$

T_{pes} jest więc analogicznie jak w przypadku pierwszego algorytmu rzędu n^2.

Analiza kosztu oczekiwanego (wymagająca trochę dłuższych rachunków) prowadzi do wyniku $T_{ave} = \frac{n^2}{4} - \frac{n}{4}$, co też jest niestety funkcją rzędu n^2. Zastosowane usprawnienie nie okazało się więc wystarczające. Poprawę oczekiwanej złożoności czasowej (przy złożoności pesymistycznej takiej jak poprzednio) uzyskujemy stosując algorytm zwany sortowaniem szybkim. Algorytm ten opiera się na następującym pomyśle. Poprzestawiajmy elementy ciągu tak, aby po lewej stronie znalazły się wszystkie elementy mniejsze od losowo wybranego elementu v, po prawej stronie wszystkie od niego większe, i niech element v oddziela obie te grupy. Otrzymamy następujący ciąg $a_1, \ldots, a_{j-1}, v, a_{j+1}, \ldots, a_n$. Opisaną wyżej operację powtarzamy teraz dla każdego z ciągów $a_1, \ldots, a_{j-1}$ oraz $a_{j+1}, \ldots, a_n$, który ma długość większą od jedności.

Algorytm szybkiego sortowania jest kolejnym przykładem algorytmu rekurencyjnego. Tego typu algorytm wraz z szerszym omówieniem zamieściliśmy w *Delcie* 12/1984.

W podanym niżej algorytmie wybieramy jako v pierwszy element podciągu. Do ciągu dodajemy dodatkowy element a_{n+1} większy od wszystkich elementów ciągu ($a_{n+1} = \infty$). Pełni on rolę „strażnika". Sprawdźcie sami, co by się stało, gdyby pierwszy element był największy w ciągu i nie byłoby takiego strażnika.



Algorytm 3: Sortowanie szybkie

```
parametry l: indeks początku sortowanego podciągu
          p: indeks końca sortowanego podciągu
v := a_i; i := l; j := p+1; {a_k < a_{p+1} dla k = l, ..., p}
dopóki i < j powtarzaj {podział na podciągi}
| dopóki a_i ⩽ v powtarzaj i := i+1; {a_i jest po dobrej stronie}
| dopóki a_j > v powtarzaj j := j-1; {a_j jest po dobrej stronie}
| jeśli i < j to x := a_i; a_i := a_j; a_j := x; {zatrzymaliśmy się
| na takich a_i oraz a_j, które nie były po dobrych stronach,
| zamieniamy je miejscami}
a_l := a_j; a_j := v {wstawiamy v między podciągi}
jeśli l < j-1 to wykonaj sortowanie szybkie startując od l = l
    i p = j;
jeśli p > j+1 to wykonaj sortowanie szybkie startując od l = j+1
    i p = p;
```

Aby posortować ciąg, należy wystartować od $l = 1$ i $p = n$.

Obok przykład działania algorytmu dla ciągu 4, 3, 1, 7, 2, 8, 6, 5. Element oznaczony * pełni rolę strażnika w przypadku, gdy a_l jest największym elementem podciągu.

W pesymistycznym przypadku (tu będzie to np. ciąg posortowany) liczba porównań będzie znowu rzędu n^2. Zauważmy, że suma wszystkich porównań wykonywanych na jednym „poziomie" jest nie większa niż n. Gdyby więc przy każdym podziale na część „lewą" i „prawą" otrzymane części były równe, to liczba porównań byłaby rzędu $n\log_2 n$. Okazuje się jednak, że w przypadku tego algorytmu oczekiwana złożoność czasowa nie odbiega wiele od tego korzystnego przypadku i jest rzędu $n\log_2 n$. Jest to więc niewątpliwy postęp w stosunku do poprzednich algorytmów.

Istnieją inne algorytmy umożliwiające posortowanie ciągu ze średnią złożonością czasową tego samego rzędu co algorytm sortowania szybkiego. Interesujące wydaje się być pytanie, czy da się to zrobić jeszcze szybciej. Okazuje się, że nie. Jeśli kolejność elementów wyznaczamy na podstawie porównywania tych elementów, nie da się poprawić rzędu funkcji oczekiwanej złożoności czasowej.

Zadanie. Sortujemy ciąg $a_1, \ldots, a_n$ w następujący sposób: generujemy wszystkie możliwe permutacje ciągu $a_1, \ldots, a_n$ i dla każdej z nich sprawdzamy, czy tworzy ona ciąg posortowany. Uzasadnić, iż średnia liczba porównań jest co najmniej $\frac{n!}{2}$. Zakładając, że maszyna wykonuje 10^5 operacji na sekundę obliczyć, ile co najmniej czasu średnio potrzeba na posortowanie ciągu długości 15 opisaną wyżej metodą, a ile za pomocą algorytmu szybkiego sortowania.

Komputer zamiast zecera

Zawody rodzą się i umierają wraz z rozwojem techniki. Nie ma już dorożkarzy, zapalaczy latarni gazowych i wielu innych zawodów. Także rozwój komputerów eliminuje niektóre grupy zawodowe. Nie każdy jednak wie, że największą taką grupą (przynajmniej w krajach wysoko rozwiniętych) są ... zecerzy.

Od czasów Gutenberga do połowy XX wieku technika druku bardzo się rozwinęła. Ręczne składanie kolejnych stron z drewnianych czcionek zastąpiono odlewaniem całych wierszy w specjalnych urządzeniach, zwanych linotypami. Ich obsługiwanie było właśnie zajęciem zecerów. Praca zecera odbywała się w bardzo trudnych warunkach. Linotypy były bardzo hałaśliwe, a pary gorącego stopu ołowiowo-antymonowego zatruwały pracujących. Nic więc dziwnego, że — gdy tylko było to możliwe — wprowadzono tzw. skład komputerowy.

Cały tekst artykułu gazetowego albo książki jest przygotowywany na komputerze. Na ekranie widać dokładny obraz tego tekstu, z uwzględnieniem różnych krojów czcionki, zmiennej szerokości znaków (porównajcie długość słów *iiiii* oraz *mmmmm*), wytłuszczeń, kroju pochyłego (kursywy) itd.

Komputer automatycznie dzieli tekst na wiersze i kolumny, a następnie zapisuje na nośniku magnetycznym (np. na dysku elastycznym — od 100 do 500 stron maszynopisu na krążku o średnicy do 20 cm i grubości dwóch okładek *Delty*). Dalsze etapy przygotowania matryc do drukowania przebiegają już bez bezpośredniego udziału człowieka.

Poza likwidacją miejsc pracy szkodliwych dla zdrowia skład komputerowy przyniósł także inne korzyści. Przy przygotowywaniu wydania gazety problemem było zawsze tzw. łamanie numeru, tzn. podział posiadanego materiału na szpalty i kolumny, rozmieszczenie, wielkość i krój tytułów, zdjęć itp. Przy składzie ręcznym wymagało to fizycznego manipulowania matrycami, składania pojedynczych wierszy w szpalty, układania kolumn ze szpalt, ręcznego układania tytułów z dużych czcionek i wielu jeszcze innych czasochłonnych zajęć. W praktyce redaktor wydania mógł wypróbować maksymalnie 3—4 różne wersje rozmieszczenia materiału na stronach. Poza tym częste były przypadki przemieszania wierszy (albo zamiany wierszy w sąsiednich szpaltach).

Przy składzie komputerowym redaktor może zażądać, aby komputer przesunął artykuł na wskazaną kolumnę, aby zmienił krój czcionki tytułu i odpowiednio przemieścił pozostałe artykuły. Wynik jest natychmiast widoczny na ekranie. Pozwala to wypróbować nawet kilkadziesiąt wariantów złamania numeru przed wyborem najlepszego.

Jeszcze większe korzyści daje skład komputerowy przy druku książek. W dotychczasowym systemie autor pisał rękopis, na który nanosił wielokrotnie poprawki, maszynistka przepisywała rękopis, a maszynopis był przekazywany do redakcji, po recenzjach na maszynopis były nanoszone poprawki i uwagi techniczne (krój czcionki itp.), wreszcie zecer składał matryce książki. Tak długa droga powodowała powstawanie licznych błędów. Wprowadzanie poprawek do już złożonego tekstu wymagało ponownego złożenia obszernego fragmentu tekstu (często pojawiały się nowe błędy), a wznowienie lub wydanie w innym formacie — na ogół złożenia całości od nowa.

Przy składzie komputerowym cały tekst książki jest przygotowywany od razu za pomocą komputera. Tekst, zapamiętany na nośnikach magnetycznych, może być łatwo i szybko modyfikowany zgodnie z sugestiami recenzentów i redakcji. Nikt poza autorem nie pisze ani nie przepisuje tekstu (o ile tekst został zapisany na komputerze), a więc nie ma obawy o powstanie błędów wynikających z niezrozumienia zawartości książki. Wprowadzanie zmian jest możliwe i łatwe jeszcze na moment przed oddaniem książki do druku. Automatyczna produkcja matryc przyspiesza także pracę nad książką w samej drukarni. A przy ponownych wydaniach wystarczy wyjąć z szuflady dysk elastyczny ...

A co z zecerami tam, gdzie powszechnie wprowadzono skład komputerowy? Część przeszła na renty i wcześniejsze emerytury, pozostali obsługują maszyny do składu komputerowego. Kilkudziesięciu pracuje w różnych muzeach techniki, w których składa się drobne ulotki i programy zwiedzania przy użyciu muzealnych już dziś linotypów.

J. D.

Od redakcji: *Delta* jest składana przy użyciu lino i monotypów oraz (w trudniejszych miejscach) ręcznie. Drukarnie, w tym i nasza, są wyposażane w urządzenia do składu komputerowego, jednak redakcje (nawet redakcja *Informatyki*) nie dysponują możliwością zapisywania tekstu na komputerze. Nie mówiąc już o autorach, którzy rzadko mają dostęp choćby do mikrokomputera, zupełnie zresztą w tej sprawie nieprzydatnego.