motywy
LOGO

O algorytmice, probabilistyce i programowaniu funkcyjnym

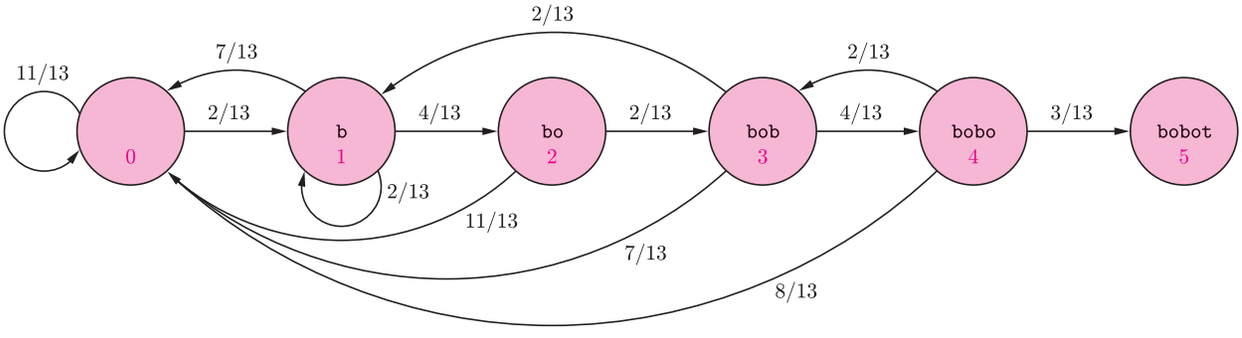
Andrzej WALAT

Pomiędzy algorytmiką i probabilistyką istnieją liczne powiązania. W wielu przypadkach, gdy nie znamy wzoru pozwalającego obliczyć dokładną wartość szukanej wielkości, posługujemy się programami, które wykorzystując komputerowe generatory liczb losowych, znajdują odpowiednio dokładne przybliżenie szukanej liczby. Z drugiej strony, istnieją też algorytmy dające dokładne, ale nie stuprocentowo pewne rozwiązania, na przykład odpowiedź tak albo nie na pytanie, czy dana liczba jest pierwsza. Czytelnikom zainteresowanym tematem *Algorytmika i probabilistyka* polecam książkę Mitzenmachera i Upfala [2]. W tym artykule przedstawię przykład dokładnego algorytmicznego rozwiązania probabilistycznego problemu.

Problem. Dane jest hasło oraz słowo utworzone z liter występujących w haśle. Będziemy losowali litery z hasła tak długo, aż otrzymamy dane słowo. Ile średnio losowań trzeba wykonać, aby utworzyć dane słowo?

Przykład. Dane hasło to słynna kwestia Hamleta `to be or not to be`. Pytamy: Ile razy średnio będziemy losować litery z hasła, by utworzyć słowo `bobot`? Teoretycznie możemy osiągnąć cel już po pięciu krokach, jeśli w pierwszych pięciu losowaniach otrzymamy litery: `b`, `o`, `b`, `o`, `t`, ale prawdopodobieństwo tak szybkiego sukcesu jest znikomo małe – równe $\frac{2}{13} \cdot \frac{4}{13} \cdot \frac{2}{13} \cdot \frac{4}{13} \cdot \frac{3}{13}$.

Średnia liczba losowań, które trzeba wykonać, aby otrzymać cel, jest znacznie większa. Jak ją obliczyć? Generalnie są dwa sposoby. Pierwszy – empiryczny, polegający na tym, że przeprowadzamy n-krotnie symulację losowania liter z hasła aż do utworzenia danego słowa i obliczamy średnią liczbę losowań. Drugi – teoretyczny. Rysujemy graf:



Rys. 1

Tworzymy odpowiedni układ równań liniowych, dla naszego przykładu jest to układ:

$$\begin{cases} x_0 = 1 + \frac{11}{13}x_0 + \frac{2}{13}x_1, \\ x_1 = 1 + \frac{7}{13}x_0 + \frac{2}{13}x_1 + \frac{4}{13}x_2, \\ x_2 = 1 + \frac{11}{13}x_0 + \frac{2}{13}x_3, \\ x_3 = 1 + \frac{7}{13}x_0 + \frac{2}{13}x_1 + \frac{4}{13}x_4, \\ x_4 = 1 + \frac{8}{13}x_0 + \frac{2}{13}x_3, \end{cases} \quad \text{równoważny z:} \quad \begin{cases} 13 - 2x_0 + 2x_1 = 0, \\ 13 + 7x_0 - 11x_1 + 4x_2 = 0, \\ 13 + 11x_0 - 13x_2 + 2x_3 = 0, \\ 13 + 7x_0 + 2x_1 - 13x_3 + 4x_4 = 0, \\ 13 + 8x_0 + 2x_3 - 13x_4 = 0. \end{cases}$$

Stany zostały ponumerowane kolorowymi cyframi.

Niewiadoma x_i w tym układzie to teoretyczna średnia liczba losowań, które trzeba wykonać, by przejść od stanu i do stanu końcowego (w naszym przykładzie 5). Rozwiązujemy układ. Wyznaczona w ten sposób wartość $x_0 = 1933{,}82$ to szukana teoretyczna średnia.

Układ, którym posłużyliśmy się w rozwiązaniu zadania, ma szczególną cechę, po pomnożeniu wszystkich równań przez długość hasła 13 i przeniesieniu wszystkich

13	−2	2			
13	7	−11	4		
13	11	0	−13	2	
13	7	2	0	−13	4
13	8	0	0	2	−13

Rys. 2. Tabela współczynników układu.

13	−2	2		
13	7	−11	4	
13	11	0	−13	2
221	123	26	0	−161

Rys. 3. Tabela współczynników układu zredukowanego.

371293	−192

Rys. 4.

wyrazów na lewą stronę, a następnie wpisaniu współczynników do tabeli, zauważymy, że tworzą one schodkową strukturę, która podpowiada sposób rozwiązania układu – metodą eliminacji kolejnych (od końca) niewiadomych.

Po wyeliminowaniu ostatniej niewiadomej przez zastąpienie dwóch ostatnich równań odpowiednią ich kombinacją otrzymujemy tabelę współczynników układu zredukowanego (rys. 3). Po kolejnych trzech krokach redukcji otrzymujemy jednowierszową tabelę (rys. 4), która reprezentuje równanie $371293 - 192x_0 = 0$. Dokładna wartość niewiadomej $x_0 = 371293/192$. Możemy teraz wyznaczyć pozostałe niewiadome, ale nie będziemy tego robili, ponieważ odgrywają one tylko pomocniczą rolę i ich wartości nas nie interesują. Przyjmijmy, że interesującym nas rozwiązaniem (pierwiastkiem) schodkowego układu równań jest wartość x_0 (a nie, jak zwykle, n-tka liczb). Czas rozwiązania schodkowego układu n-równań liniowych jest proporcjonalny do n^2, podczas gdy czas rozwiązania dowolnego układu równań liniowych metodą Gaussa–Jordana jest proporcjonalny do n^3.

Rozwiązanie problemu sformułowanego na wstępie w ogólnym przypadku, czyli teoretyczna oczekiwana (średnia) liczba losowań liter z danego hasła, które trzeba wykonać, by otrzymać dane słowo docelowe, to pierwiastek schodkowego układu równań zdeterminowanego przez dane hasło i docelowe słowo. Tę ogólną ideę możemy zapisać formalnie w postaci następującej definicji w Logo:

```
oto olilo :hasło :cel
wynik pierwiastek sur :hasło :cel
już
```

Różni się ona od sformułowania w naturalnym języku drobnymi szczegółami notacyjnymi i tym, że zamiast pełnych sformułowań: „oczekiwana liczba losowań” oraz „schodkowy układ równań” używamy skrótów `olilo` oraz `sur`. Faktycznie `olilo` jest funkcją, która dla danych dwóch słów wyznacza wynik liczbowy. Jest ona złożeniem dwóch bardziej elementarnych funkcji: funkcji `pierwiastek`, która mając dany dowolny schodkowy układ równań liniowych, znajduje jego rozwiązanie, z funkcją `sur`, która dla danych dwóch słów (hasła i celu) generuje odpowiedni schodkowy układ równań. Żeby posłużyć się funkcją `olilo`, musimy jeszcze zdefiniować funkcje `pierwiastek` oraz `sur`. Ale przedtem kilka słów o ważnej idei programowania.

Przez wiele lat w informatyce zdecydowanie dominowały języki *programowania imperatywnego*. Z czasem zaczęły zyskiwać znaczenie inne metody i języki programowania: *programowanie funkcyjne*, *programowanie w języku logiki* oraz *programowanie zorientowane obiektowo*. Programowanie imperatywne jest nieodłącznie związane z tzw. *myśleniem operacyjnym*. Kiedy tworzymy lub analizujemy programy i chcemy je rozumieć, myślimy o ich wykonawcy – o rzeczywistym komputerze z jego rejestrami, czyli komórkami, w których zapisujemy słowa zerojedynkowe albo o wirtualnej maszynie z jej zmiennymi, które mają wartości określonych typów. Komputerowa realizacja każdego zadania algorytmicznego redukuje się w końcu do odczytywania wartości zmiennych, wykonywania na nich jakichś operacji elementarnych i zapamiętywania wyników jako nowych wartości innych zmiennych (przypisywania ich innym zmiennym). Stąd tak wielkie znaczenie instrukcji przypisania, że jej symbol (dwuznak :=) stał się wręcz symbolem całej informatyki, a przynajmniej symbolem programowania. Programowanie funkcyjne oznacza zasadniczą zmianę w stosunku do myślenia operacyjnego. Kiedy tworzymy programy lub odczytujemy ich znaczenie, nie myślimy o ich wykonawcy – realnej lub wirtualnej maszynie – tylko o zależności funkcyjnej wyniku od danych. Jeśli jest to zależność nieelementarna, próbujemy rozłożyć

ją na funkcje bardziej elementarne (przedstawić ją jako superpozycję bardziej elementarnych funkcji). Tak właśnie zdefiniowaliśmy funkcję `ololo`. W programowaniu funkcyjnym zasadniczo nie posługujemy się instrukcją przypisania. Ambitnym Czytelnikom polecam obszerną monografię poświęconą programowaniu funkcyjnemu [1].

Teraz zdefiniujemy funkcję `pierwiastek`. Jej daną jest schodkowy układ równań, ale w jakiej postaci? W Logo złożone struktury danych reprezentujemy w postaci list. Schodkowe układy równań będziemy reprezentowali jako listy kolejnych wierszy współczynników. Układ przedstawiony na rysunku 2 reprezentuje następująca lista list:

`[[13 8 0 0 2 -13] [13 7 2 0 -13 4] [13 11 0 -13 2] [13 7 -11 4] [13 -2 2]]`.

Układ składający się z jednego tylko równania pierwszego stopnia przedstawiony na rysunku 4 reprezentuje lista, której jedynym elementem jest lista dwóch liczb:

`[[371293 -192]]`.

Pierwiastek danego schodkowego układu równań obliczamy w następujący sposób. Jeśli układ jest elementarny – jest tylko jedno równanie reprezentowane przez listę postaci `[[a b]]`, gdzie `a` oraz `b` to dwie liczby całkowite, wynikiem jest ułamek reprezentowany przez listę dwóch liczb `[a -b]`. W przeciwnym przypadku wynikiem jest pierwiastek reduktu danego układu równań.

```
oto pierwiastek :sur
jeśli długość :sur = 1 [wynik lista pierw pierw :sur -1 * ost pierw
                                                              :sur]
wynik pierwiastek redukt :sur
już
```

Redukt nieelementarnego schodkowego układu równań wyznaczamy, zastępując pierwsze dwa równania (wiersze współczynników) ich odpowiednią kombinacją.

```
oto redukt :sur
wynik nap kombinacjaRównań element 1 :sur element 2 :sur bp bp :sur
już
```

Kombinację dwóch równań reprezentowanych przez dwie równej długości listy współczynników wyznaczamy w następujący sposób: od iloczynu pierwszego równania przez ostatni współczynnik drugiego odejmujemy iloczyn drugiego równania przez ostatni współczynnik pierwszego i ta różnica bez ostatniego elementu jest wynikiem.

```
oto kombinacjaRównań :r1 :r2
wynik bo ((ost :r2) * :r1 -- (ost :r1) * :r2)
już
```

W żadnej z powyższych trzech definicji nie użyliśmy instrukcji przypisania. Teraz możemy już użyć funkcji `pierwiastek` i sprawdzić poprawność naszych definicji. Po napisaniu polecenia

```
pokaż pierwiastek [[13 8 0 0 2 -13] [13 7 2 0 -13 4] [13 11 0 -13 2]
[13 7 -11 4] [13 -2 2]]
```

komputer wypisał na ekranie tekstowym: `[371293 192]`. Jest to dokładna wartość szukanego pierwiastka w postaci pary [licznik mianownik]. Ten test wypadł pomyślnie, ale by rozwiązanie problemu sformułowanego na wstępie było kompletne, trzeba jeszcze zdefiniować funkcję `sur`. Zdefiniowanie tej funkcji pozostawiam Czytelnikowi. Kompletne rozwiązanie zadania można znaleźć na stronie internetowej *Delty*.

Literatura

[1] H. Abelson, G.J. Sussman, J. Sussman, *Struktura i interpretacja programów komputerowych*, WNT Warszawa 2002.

[2] M. Mitzenmacher, E. Upfal, *Probability and Computing. Randomized Algorithms and Probabilistic Analysis*, Cambridge University Press 2005.