Krótko o skracaniu

*Tomasz KAZANA**

* Instytut Informatyki, Uniwersytet Warszawski

Załóżmy, że chcemy zapisać w pliku długi tekst składający się wyłącznie z liter A, B, C i D. Przez plik będziemy w tym artykule rozumieli dowolny ciąg bitów, to znaczy ciąg przeplatających się zer i jedynek.

Najprostszy i chyba naturalnie nasuwający się pomysł jest następujący: umówmy się, że literę „A” będziemy oznaczać na przykład jako `00`, literę „B” jako `01`, „C” jako `10`, a „D” jako `11`. Oczywiście, ponieważ jedna litera wymaga do zapisu dwóch bitów, to tak zapisany tekst zajmie $2n$ bitów, gdy liter było n. Czy da się lepiej?

W ogólności (to jest wcale nieoczywiste) nie, ale w pewnych szczególnych okolicznościach – jak najbardziej. Na przykład rozważmy sytuację, gdy litery nie występują równie często. Powiedzmy, że litera A znajduje się na aż 85% pozycji całego tekstu, B – na 6%, C – na 5%, a D – na 4% pozycji. Przy takim *rozkładzie* literek lepszy efekt da następujące *kodowanie*:

$$\mathrm{A} \equiv \texttt{0},\ \mathrm{B} \equiv \texttt{10},\ \mathrm{C} \equiv \texttt{110},\ \mathrm{D} \equiv \texttt{111},$$

gdyż wynikowy plik zajmie tym razem:

$$0{,}85n \cdot 1 + 0{,}06n \cdot 2 + 0{,}05n \cdot 3 + 0{,}04n \cdot 3 = 1{,}24n$$

bitów.

Metadane

Gwoli ścisłości – plik opisany wyżej w praktyce zajmie zwykle nieco więcej niż te wyliczone właśnie $1{,}24n$. Dlaczego? Otóż, aby możliwe było odczytanie pliku, gdzieś musi być zapisana informacja o szczegółach wybranego kodowania. Możemy sobie wyobrazić, że jest ono znane z góry (bo na przykład rozkład literek wynika z jakiegoś zjawiska fizycznego, którego przebieg zapisujemy do pliku) i wówczas jest ono specyficzną cechą przyjętego na stałe formatu naszego pliku.

Częściej jednak jest inaczej – zapisujemy jakąś informację do pliku, a wyboru kodowania chcemy dokonywać „na gorąco”, na podstawie oceny aktualnych danych. Wówczas jednak format pliku musi uwzględniać, że to sam plik opisuje najpierw sposób kodowania, a dopiero potem – zawiera już informację właściwą, opisaną za pomocą deklarowanego kodowania. W takim przypadku wszystkie informacje poboczne, a więc te niebezpośrednio opisujące treść pliku, jak chociażby opis kodowania literek w naszym przykładzie, nazywamy *metadanymi* i również one muszą zostać uwzględnione w wyliczeniach rozmiaru pliku wynikowego.

Bardziej konkretnie, nasze przykładowe kodowanie mogłyby opisywać na przykład takie metadane:

$$\texttt{000111000111110001111111},$$

które skonstruowaliśmy w następujący sposób: każde słowo kodowe (np. `110`) jest zapisane poprzez zdublowanie każdego swojego bitu (w tym przypadku: `111100`), a rolę przecinka oddzielającego kolejne słowa kodowe odgrywa sekwencja `01`. Łatwo zauważyć, że tak pomyślane metadane daje się jednoznacznie odczytać, w naszym przypadku do oczekiwanego komunikatu `0,10,110,111,` (z przecinkiem na końcu).

Ostatecznie, uwzględniając uwagi o metadanych, np. zapis komunikatu „AABAAACADA” przy metodzie opisanej wyżej wyglądałby następująco:

$$\texttt{0001110001111100011111110100100001100111}0.$$

Ogólnie: zapis komunikatu o n literkach w opisany sposób, przy rozkładzie literek $(85\%, 6\%, 5\%, 4\%)$, będzie zajmował dokładnie $(26 + 1{,}24n)$ bitów.

Bezprefiksowość

Podany przykład przekonuje nas, że – jeśli chcemy pliki zapisywać skrótowo, czyli, używając fachowej terminologii, je kompresować – warto rozważyć dobranie specyficznego kodowania literek do ich rozkładu. Takie podejście

Rozwiązanie zadania F 1061.
Na powierzchni przewodnika umieszczonego w polu elektrycznym wytwarza się taki rozkład ładunku, że wewnątrz przewodnika natężenie wypadkowego pola (sumy pola zewnętrznego i pola wytworzonego przez rozkład ładunków na powierzchni) wynosi dokładnie zero. Oznacza to, że ładunki na powierzchni są źródłem jednorodnego pola $-\vec{E}$ w całej objętości kuli. Energia $\mathcal{E}$ tego pola wynosi

$$\mathcal{E} = \frac{1}{2}\varepsilon_0 E^2 V,$$

gdzie ε_0 jest przenikalnością elektryczną próżni, a V oznacza objętość kuli (obszar występowania pola $-\vec{E}$). Po wyłączeniu zewnętrznego pola ładunki „wracają” w położenia, jakie miały przed włączeniem tego pola. Ich przepływowi towarzyszy wydzielanie ciepła Q w ilości równej energii pola $\mathcal{E}$. W przypadku kuli o promieniu $2R$ pole $-\vec{E}$ wypełnia objętość 8 razy większą, a więc po wyłączeniu pola wydzieli się ilość ciepła równa $8Q$.

zawiera jednak pewne ryzyko, nad którym prześlizgnęliśmy się w opisie wyżej. Otóż, jeśli różne literki są opisywane słowami kodowymi o różnej długości, to nie mamy pewności co do jednoznaczności odczytu! Spójrzmy chociażby, co by się stało, gdybyśmy wybrali nieco inne kodowanie (względem pierwotnego przykładu zmieniliśmy tutaj tylko jeden bit w słowie kodowym dla litery „B”):

$$\mathrm{A} \equiv \texttt{0},\ \mathrm{B} \equiv \texttt{11},\ \mathrm{C} \equiv \texttt{110},\ \mathrm{D} \equiv \texttt{111}.$$

Przy takim kodowaniu plik (pomijamy metadane) `110111111` mógłby mieć cztery różne interpretacje: BABBB, BADD, CBBB oraz CDD...

Czytelnik proszony jest o spróbowanie formalnego pokazania, że istotnie każde bezprefiksowe kodowanie jest jednoznaczne w odczycie.

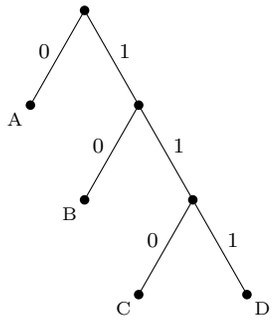
Musimy więc zachowywać dużą ostrożność i słowa kodowane dobierać tak, aby niejednoznaczność odczytu nie była możliwa. Na szczęście okazuje się, że aby to osiągnąć, wystarczy (nie jest to warunek konieczny, ale wystarczający i również dość wygodny) upewnić się, że zbiór proponowanych słów kodowych jest *bezprefiksowy*. To znaczy, że żadne słowo kodowe nie jest prefiksem żadnego innego. Widać, że kodowanie z początku tekstu ma taką własność, ale to po modyfikacji już nie – chociażby słowo kodowe dla B jest prefiksem słowa kodowego dla C.

O co zapyta rasowy matematyk?

Gdy już wiemy (bo mamy przykład), że czasem dobranie kodowania do rozkładu literek *opłaci się*, to powstaje naturalne pytanie: jak szukać w takim razie optymalnego kodowania, powiedzmy, przy założeniu, że ograniczamy się do kodowań bezprefiksowych (można pokazać, że to ograniczenie nie wpływa na efektywność ostatecznego rozwiązania).

Okazuje się, że ten problem jest już rozwiązany, a elegancka konstrukcja pozwalająca zawsze znaleźć najlepsze rozwiązanie nazywa się *kodowaniem Huffmana*. Zanim przejdziemy do jego ekspozycji, wprowadźmy jeszcze pewien wygodny graficzny sposób opisu kodowania bezprefiksowego. Otóż zbiór słów kodowych będziemy reprezentować poprzez drzewo (tzw. *drzewo kodowe*), w którego liściach będą zapisane literki, a ścieżka z korzenia do liści będzie przedstawiać wybrane słowo kodowe. Reguła jest prosta – skręt w lewo to `0`, a w prawo – to `1`. Drzewko dla kodu, który analizujemy od początku tego artykułu, będzie więc wyglądało jak na rysunku 1.

Anegdota głosi, że David Huffman rozwiązał opisywany problem w ramach studenckiej pracy domowej, w której jedno z zadań było problemem otwartym.



Rys. 1

Mając tę notację, możemy już łatwo opisać algorytm budowania kodu Huffmana:

1. Niech X będzie strukturą przechowującą pary: (drzewko kodowe; liczba rzeczywista);
2. X zainicjuj następująco: dla każdej literki L o prawdopodobieństwie wystąpienia p włóż do X parę: (pojedynczy wierzchołek z etykietą L; p);
3. Dopóki X zawiera co najmniej dwa elementy, wykonuj w pętli:
   - wybierz (i usuń) z X dwa elementy o najmniejszej drugiej współrzędnej – oznaczmy je przez (D_1, p_1) oraz (D_2, p_2);
   - dodaj do X nowy element $\left(\begin{smallmatrix} & 0 & & 1 & \\ \mathrm{D}_1 & & & & \mathrm{D}_2 \end{smallmatrix},\ p_1 + p_2\right)$.
4. Gdy X ma już tylko jeden element – zwróć go jako wynik ustalania optymalnego kodowania.

Dla czytelności obrazu przedstawmy jeszcze krok po kroku zawartość struktury X w algorytmie Huffmana dla rozważanego przez nas przypadku:

$$(\dot{A}, 0{,}85), (\dot{B}, 0{,}06), (\dot{C}, 0{,}05), (\dot{D}, 0{,}04) \Rightarrow$$

$$\Rightarrow (\dot{A}, 0{,}85), (\dot{B}, 0{,}06), \left(\begin{smallmatrix} & 0 & & 1 & \\ \mathrm{C} & & & & \mathrm{D} \end{smallmatrix},\ 0{,}09\right) \Rightarrow$$

$$\Rightarrow (\dot{A}, 0{,}85), \left(\begin{smallmatrix} & 0 & & 1 & \\ \mathrm{B} & & 0 & & 1 \\ & & \mathrm{C} & & \mathrm{D} \end{smallmatrix},\ 0{,}15\right) \Rightarrow \left(\begin{smallmatrix} & & 0 & & 1 & \\ & \mathrm{A} & & 0 & & 1 \\ & & \mathrm{B} & & 0 & & 1 \\ & & & & \mathrm{C} & & \mathrm{D} \end{smallmatrix},\ 1\right).$$

Dowód faktu (wcale nie taki trudny!), że podana konstrukcja zwraca wynik optymalny, pozostawimy Czytelnikowi. Napiszmy tylko, że uzyskany w ten

Aby zbliżyć się do granicy teoretycznej dowolnie blisko, stosuje się pewne sztuczki, np. tak zwane kodowanie Shannona–Fano, ten temat niestety wykracza poza ramy naszego artykułu. Można go w wielkim uproszczeniu opisać jako zmianę alfabetu poprzez sklejanie literek. A więc – zamiast rozważania alfabetu (A, B, C, D), można analizować (AA, AB, AC, AD, BA, BB, BC, BD, CA, CB, CC, CD, DA, DB, DC, DD) z odpowiednio dobranym rozkładem. Wówczas metadane opisu kodu będą rosły, ale sam tekst właściwy – dzięki dobraniu kodu Huffmana do „lepszych" danych – może okazać się krótszy niż dla klasycznego kodowania Huffmana dla alfabetu (A, B, C, D).

Dość arbitralne stwierdzenie, że losowych danych nie da się kompresować, może zaintrygować. Czytelnik Złapany proszony jest o dalsze poszukiwania, np. podążając w stronę pojęcia złożoności Kołmogorowa.

sposób stopień kompresji ma silny związek z pojęciem entropii Shannona. Da się bowiem pokazać, że średnia długość słowa kodowego w rozważanym problemie nie może być nigdy mniejsza niż entropia rozkładu literek. Nie dla każdego zestawu danych da się ją uzyskać dokładnie, ale kod Huffmana daje zawsze maksimum tego, co daje się wycisnąć dla danego zbioru i rozkładu literek. Co więcej, daje się pokazać, że ten rezultat jest co najwyżej o 1 gorszy od teoretycznego dolnego oszacowania wynikającego z teorii entropii Shannona.

Jeszcze inne okoliczności

Zasadniczo losowych danych nie da się kompresować. Aby więc cokolwiek udało się zapisać krócej, musimy znaleźć jakieś sprzyjające okoliczności. Wyżej eksploatowaliśmy okoliczność niejednostajnego rozkładu literek. Oczywiście mogą wystąpić także inne.

Na przykład może się zdarzyć, że rozkład literek jest co prawda ogólnie jednostajny, ale już niekoniecznie w pewnych fragmentach. I mogłoby się też zdarzyć, że w pierwszej ćwiartce danych rozkład wynosi $(85\%, 6\%, 5\%, 4\%)$, w drugiej – $(4\%, 85\%, 6\%, 5\%)$, w trzeciej – $(5\%, 4\%, 85\%, 6\%)$, a w czwartej – $(6\%, 5\%, 4\%, 85\%)$. Wówczas widać, że nie opłaca się ustalić jednego kodowania literek w całym pliku, jednak można/warto zmienić go kilka razy. Bardziej trzeba się napracować, aby wymyślić odpowiedni opis metadanych dla takiego przypadku, ale na pewno ostateczny rozrachunek będzie korzystny.

Zupełnie inne jakościowo zjawisko występuje w danych, w których pojawiają się czasem te same, bardzo długie bloki. Na przykład rozważmy tekst książki o taktyce piłkarskiej. W takiej pozycji zapewne frazy „Real Madryt", „Tiki-Taka", „Zinedine Zidane" czy „Gegenpressing" wystąpią wyjątkowo często. Może się więc *opłacić* ustalić (w ramach metadanych) dla tego typu częstych fraz specjalne kodowanie, które sprawi, że nie będziemy musieli często powtarzać tych samych długich określeń.

To jak w końcu działają programy do kompresji?

Współczesne programy do kompresji (takie jak ZIP) są rodzajami kombajnów wyposażonych w wiele różnorodnych algorytmów, przygotowanych na różne potencjalnie sprzyjające kompresji okoliczności. Samo kodowanie zawiera istotną i dość skomplikowaną strukturę metadanych – muszą one wszak opisywać wybór konkretnej metody z dostępnej palety (być może różny dla różnych fragmentów) oraz dodatkowe metadane wewnętrzne, specyficzne dla każdej ustalonej metody. W tym artykule podjęliśmy próbę naszkicowania takiego rozumowania na przykładzie jednej konkretnej, jak to określiliśmy, okoliczności sprzyjającej. Czytelnik Zainteresowany Tematem znajdzie dużo ciekawych informacji w Wikipedii i innych źródłach, które dość szczegółowo opisują zarówno format metadanych, jak i konkretne algorytmy z palety dostępnych możliwości.

Podkreślmy jednocześnie, że – wobec powyższego – dobry program do kompresji nie tylko posiada bogaty zbiór dostępnych narzędzi: musi dysponować również wyrafinowanym algorytmem, który jest w stanie duże dane szybko przeanalizować (metodami stochastycznymi, ale nie tylko) oraz dobrać do nich skuteczną strategię wyboru narzędzi, opisać ją poprzez metadane i ostatecznie zakodować. Nie jest to bynajmniej zadanie proste, a jednak oryginalne pomysły wciąż są możliwe!

A może coś pomińmy?

W tekście o kompresji wypada dodać coś (choć nie jest to kompletnie tematem tego artykułu) o tak zwanej kompresji stratnej. Otóż zauważono, że dla niektórych danych efekt kompresji można uzyskać nie dzięki zwięzłości zapisu poprzez sprytne kodowanie – czasem godzimy się po prostu na usunięcie części danych. Dzieje się tak, gdy dobrze wiemy, co opisują dane i które informacje są drugorzędne. W plikach muzycznych możemy na przykład wycinać niesłyszalne dla człowieka częstotliwości dźwięku, w grafice możemy próbować zmniejszać paletę barw albo rozmywać fragmenty, których „oko nie zauważy". Te metody wykraczają poza informatykę teoretyczną, gdyż mają swoje korzenie w fizjologii człowieka. Są natomiast bardzo skuteczne, a ich efektem są chociażby formaty plików MP3 czy JPG.

Rozwiązanie zadania F 1062.
Niech położenia gwiazd opisują wektory $\vec{r}_1$ i $\vec{r}_2$. Wektor łączący ich środki $\vec{R} = \vec{r}_2 - \vec{r}_1$. Gwiazdy przyciągają się wzajemnie siłą grawitacji o wartości:

$$F = \frac{Gm_1m_2}{|\vec{r}_1 - \vec{r}_2|^2} = \frac{Gm_1m_2}{R^2}.$$

Równania ruchu mają postać:

$$m_1\frac{d^2\vec{r}_1}{dt^2} = F\frac{\vec{R}}{R},$$
$$m_2\frac{d^2\vec{r}_1}{dt^2} = -F\frac{\vec{R}}{R}.$$

Dla wektora $\vec{R}$ łączącego ich środki otrzymujemy równanie ruchu:

$$\frac{d^2\vec{R}}{dt^2} = -G(m_1+m_2)\frac{\vec{R}}{R^3}.$$

Jest to równanie ruchu punktu materialnego w polu grawitacyjnym masy $m_1 + m_1$. Na podstawie prawa Keplera możemy więc znaleźć wartość odległości między gwiazdami:

$$R = \left(\frac{G(m_1+m_2)T^2}{4\pi^2}\right)^{1/3}.$$

Ściśle biorąc, powyższy wzór określa wartość długiej półosi orbity, tj. największą odległość między środkami mas gwiazd w ich ruchu względnym.