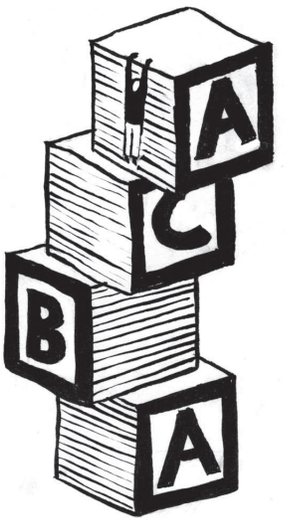
Kwadraty

Jakub RADOSZEWSKI

Tym razem zajmiemy się trochę innymi kwadratami niż zazwyczaj. Chodzi mianowicie o napisy postaci x^2, czyli sklejenie jakiegoś słowa (ciągu liter) x z nim samym. Przykładowymi kwadratami występującymi w języku polskim są słowa *mama*, *kankan*, *rowerowe*, *wałowało*, *esemesem*.

Jeśli rozważamy jakieś słowo, może nas interesować, czy jest ono kwadratem, ale także czy jakieś jego podsłowo (tzn. spójny fragment) jest kwadratem. Jeżeli nie, to słowo takie nazywamy *bezkwadratowym*. Zastanówmy się przez chwilę nad tym, jak konstruować słowa bezkwadratowe.



Najprościej wybrać słowo, w którym wszystkie litery są różne, np. `abcde...` Gdyby liter w alfabecie było nieskończenie wiele, to moglibyśmy w ten sposób skonstruować dowolnie długie słowo bezkwadratowe. Nie da się ukryć, że nie jest to zbyt ciekawy przykład. No to może spróbujmy wygenerować długie słowo bezkwadratowe nad jakimś mniejszym alfabetem?

Alfabet jednoliterowy na pewno nam nie pomoże. Załóżmy więc, że mamy do dyspozycji dwie litery, `a` i `b`. Widać, że każde dwie kolejne litery w słowie bezkwadratowym muszą być różne, a zatem nasze słowo musi zaczynać się jakoś tak: `aba...` lub `bab...` Niestety, w obu tych przykładach nie możemy dołożyć już żadnej litery, gdyż wówczas otrzymamy kwadrat $(\texttt{ab})^2$ lub odpowiednio $(\texttt{ba})^2$. To oznacza, że najdłuższe słowo bezkwadratowe nad alfabetem dwuliterowym ma tylko trzy litery.

Kolejna próba: alfabet trzyliterowy. Znów konstruujemy słowo, biorąc zawsze kolejną literę różną od poprzedniej. Daje to zawsze dwie możliwości wyboru. W takim razie dodajmy warunek, że każda kolejna litera musi być różna od *środkowej* litery dotychczasowego słowa – dokładniej, przy wyznaczaniu i-tej litery interesuje nas litera o indeksie $\left\lfloor \frac{i}{2} \right\rfloor$, przy czym litery słowa numerujemy od zera. Jeżeli to kryterium wciąż dopuszcza dwie możliwości, to wybieramy tę spośród niezabronionych liter, która występuje wcześniej w alfabecie. W ten sposób otrzymujemy takie oto słowo:

`abacbcabacabcbacbcabcbacabacbcabacabcbacabacbcabcb...`

Okazuje się, że dowolnie długie słowo wygenerowane w ten sposób jest bezkwadratowe. Można ten fakt udowodnić formalnie, jednak w tym artykule zastosujemy podejście informatyczne natury eksperymentalnej: weźmiemy odpowiednio długie słowo tej postaci (np. złożone z miliona liter) i sprawdzimy za pomocą programu komputerowego, czy jest w tym słowie jakieś kwadratowe podsłowo. Jeśli okaże się, że nie, to *zapewne* wszystkie takie słowa są bezkwadratowe...

Dla Czytelnika, który zechce spróbować zmierzyć się z próbą dowodu bezkwadratowości rozważanych słów, mamy drobną wskazówkę: warto zacząć od sprawdzenia, jaki jest związek między kolejnymi literami rozważanych słów a zapisami binarnymi odpowiadających im pozycji.

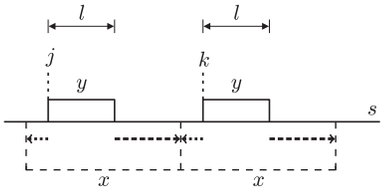
Nie jest wcale łatwo zaproponować efektywny, a zarazem nieskomplikowany algorytm sprawdzający, czy dane słowo jest bezkwadratowe – zachęcamy Czytelnika do próby samodzielnego zmierzenia się z tym problemem. Poniżej przedstawiamy elegancki algorytm o złożoności czasowej $O(n \log n)$, przy czym n to długość badanego słowa s, wzorowany na trudno dostępnej i trochę zapomnianej pracy M. Maina i R. Lorentza sprzed 25 lat.

Zacznijmy od prostego sprawdzenia, czy w słowie s znajduje się jakaś para równych kolejnych liter – to eliminuje nam kwadraty słów długości 1. W głównej części algorytmu wykonujemy $O(\log n)$ kroków; w i-tym kroku (dla $i = 1, 2, 3, \ldots$) sprawdzamy, czy słowo s zawiera podsłowo kwadratowe x^2, takie że długość x (oznaczenie: $|x|$) należy do przedziału domknięto-otwartego $[2^i, 2^{i+1})$. Wykonując taki krok, zakładamy, że s nie zawiera podsłów kwadratowych o długości połówki krótszej niż rozważane w tym kroku. Naszym celem jest wykonanie każdego kroku w złożoności czasowej $O(n)$.

Poszukiwania żądanego kwadratu rozpoczynamy od podziału słowa s na bloki długości $l = 2^{i-1}$ (jeżeli nie dzieli się równo, to końcowej, krótszej grupy liter nie rozpatrujemy). Zauważmy, że jeżeli w s występuje kwadrat x^2, taki że $|x| \geqslant 2l$, to pierwsze wystąpienie x w ramach s musi zawierać co najmniej jeden z bloków podziału. Oznaczmy ten blok przez $s[j..j+l-1]$. To samo podsłowo pojawia się także na pozycji $j + |x|$ słowa s, choć to drugie wystąpienie nie musi już być blokiem podziału.

W naszym algorytmie rozważamy każdy kolejny blok $y = s[j..j+l-1]$ i poszukujemy wszystkich jego wystąpień w s zaczynających się na pozycjach z przedziału $[j + 2l, j + 4l)$. Co ciekawe, takie wystąpienia mogą być co najwyżej *dwa*. Faktycznie, żadne dwa wystąpienia y w ramach s nie mogą na siebie nachodzić ani nawet się stykać, gdyż wówczas wyznaczałyby one kwadrat słowa o długości nie większej niż l (dlaczego?). Stanowiłoby to sprzeczność z założeniem, że s nie zawiera kwadratu krótszego niż $2l$.

Rozwiązanie zadania M 1307.
Każdą liczbę zapiszmy w postaci $2^k m$, gdzie m jest nieparzyste. Wówczas m może przyjąć wartość $1, 3, 5, \ldots, 2n-1$ – łącznie n różnych wartości. W takim razie, wśród dowolnych $n+1$ liczb znajdą się takie dwie, które mają ten sam czynnik nieparzysty, spełniają więc warunek zadania.



Dla każdego wystąpienia y w s na pozycji $k \in [j+2l, j+4l)$ musimy jakoś sprawdzić, czy wystąpienia z pozycji j oraz k wyznaczają jakiś kwadrat x^2, taki że $|x| = k-j$. Poszukując takiego kwadratu, wystarczy skupić się na badaniu równości par liter słowa s o indeksach oddalonych o $k-j$. Najpierw sprawdzamy, czy $s[j-1] = s[k-1]$, $s[j-2] = s[k-2]$ i tak dalej, aż natrafimy na parę różnych liter albo aż dalszym indeksem dojdziemy do pozycji $j+l-1$, co oznacza, że znaleźliśmy kwadrat. Następnie powtarzamy to postępowanie, ale tym razem idąc do przodu, tzn. sprawdzamy, jak długo zachodzi $s[j+m] = s[k+m]$ dla $m = l, l+1, l+2, \ldots$ Tym razem możemy zatrzymać się, jeśli liczba wykonanych tutaj kroków powiększona o liczbę kroków wykonanych wcześniej jest nie mniejsza niż $k-j-l$, patrz rysunek. Jeżeli nie dojdziemy do wartości $k-j-l$, to łatwo zauważyć, że rozważana para wystąpień podsłowa y nie wyznacza kwadratu.

Na tym rozumowaniu oparty jest poniższy pseudokod algorytmu wykrywania kwadratu w słowie $s = s[0..n-1]$.

```
function CzyJestKwadrat(s, n)
  for i := 0 to n − 2 do
    if s[i] = s[i + 1] then return true;
  l := 1;
  while 2l ⩽ n/2 do
    j := 0;
    while j < n − l do
      Z := wystąpienia s[j..j + l − 1] zaczynające się
           na pozycjach z przedziału [j + 2l, j + 4l);
      for each k ∈ Z do
        lewo := długość najdłuższego wspólnego sufiksu
                słów s[0..j − 1] i s[0..k − 1];
        prawo := długość najdłuższego wspólnego prefiksu
                 słów s[j + l..n − 1] i s[k + l..n − 1];
        if lewo + prawo ⩾ k − j − l then return true;
      j := j + l;
    l := 2l;
  return false;
end function
```

Prefiksem słowa nazywamy dowolny jego początkowy fragment, a *sufiksem* – dowolny jego końcowy fragment.

Zastanówmy się nad złożonością czasową tego algorytmu, przy okazji uzupełniając szczegóły techniczne jego implementacji. Pierwszym interesującym miejscem jest wyznaczanie zbioru Z. Znane są różne efektywne algorytmy wyszukiwania wzorca (u nas jest to słowo y) w tekście (u nas: zadany fragment słowa s), np. algorytmy Knutha–Morrisa–Pratta, Boyera–Moore'a itp. W tym miejscu czeka nas jednak kolejne zaskoczenie: otóż w naszym programie w ogóle nie musimy używać żadnego z tych wysublimowanych algorytmów! Zaczynamy od sprawdzenia, literka po literce, czy y pasuje do s od pozycji $j+2l$. Jak w pewnym momencie zakończymy to sprawdzanie (albo znajdując wystąpienie y, albo wskutek natrafienia na parę różnych liter na odpowiadających pozycjach), to kolejną próbę przypasowania słowa y wykonujemy od pierwszej pozycji w s następującej *za* wszystkimi przejrzanymi. Faktycznie, wystąpienie słowa y w s nie może nachodzić na żadne inne wystąpienie niepustego prefiksu słowa y w s, gdyż wówczas s zawierałoby kwadrat jakiegoś prefiksu słowa y, a przecież $|y| \leqslant l$. W ten sposób znajdujemy szukane co najwyżej dwa elementy zbioru Z w czasie $O(l)$.

Skondensowany przegląd efektywnych algorytmów tekstowych zawiera artykuł W. Ryttera w *Delcie* 1/2008.

Kolejny ciekawy moment to wyznaczanie wartości *lewo* i *prawo*, wykonywane troszkę inaczej niż w opisie słownym algorytmu. Uzasadnienie poprawności pomijamy, przyjrzyjmy się kwestii złożoności czasowej. Łączna liczba operacji wykonywanych tutaj może być całkiem duża. Zauważmy jednak, że jeśli przy wyznaczaniu wspólnego sufiksu i prefiksu wykonamy więcej niż $k-j-l$ operacji, to na pewno zaraz potem zakończymy działanie algorytmu, więc możemy sobie ten jeden raz pozwolić na wykonanie nawet i rzędu $O(n)$ operacji. W przeciwnym razie liczba tych operacji nie przekroczy $k-j-l$, która to wartość – przypomnijmy – jest nie większa niż $3l-1$, czyli jest rzędu $O(l)$.

Widzimy zatem, że wnętrze wewnętrznej pętli **while** wykonujemy – poza ewentualnie jedynym jej obrotem, kończącym cały algorytm – w czasie $O(l)$. Pętla ta wykonuje $O\left(\frac{n}{l}\right)$ obrotów, co pokazuje, że koszt czasowy jednego obrotu zewnętrznej pętli **while** to $O(n)$. Ta, z kolei, wykonuje co najwyżej $O(\log n)$ obrotów, skąd wnosimy, że rzeczywiście opisany algorytm ma złożoność czasową $O(n \log n)$.

Na koniec pytanie do Czytelnika: czy można ten algorytm jakoś łatwo przerobić, tak aby wykrywał *wszystkie* kwadraty w słowie?

Rozwiązanie zadania F 784.
Aby kulka doleciała do ziemi, wystarczy, że w chwili zetknięcia się z ziemią prędkość kulki będzie równa zeru. Z zasady zachowania energii:

$$mg(r-d) - \frac{q^2}{4\pi\varepsilon_0 d} + \frac{mv_{\min}^2}{2} = -\frac{q^2}{4\pi\varepsilon_0 r}$$

otrzymujemy, zaniedbując poprawki rzędu $\frac{d}{r}$:

$$v_{\min} = \sqrt{2gr}\left(\frac{q}{\sqrt{4\pi\varepsilon_0 drmg}} - 1\right),$$

przyjmując, że $\frac{q^2}{4\pi\varepsilon_0 d^2} > mg$; w przeciwnym przypadku $v_{\min} = 0$.