Wyszukiwanie wzorca w tekście

Dane Dwa ciągi znaków (tablice znaków): wzorzec `P` długości m i tekst `T` długości n.

Wynik pozycje wszystkich wystąpień wzorca w tekście, również uwzględniając "zazębienia" wzorca samego z sobą.

```
T:  abababab    P:  abab

      abab
    abababab
    abab
```

Uwaga: tablice z wzorcem i tekstem są indeksowane od pozycji 1.

Algorytm naiwny

```
Naiwny(P,T)
// P - wzorzec, tablica [1..m],
// T - tekst, tablica [1..n],
  for s=0 to n-m
      if P[1..m]==T[s+1 .. s+m]   //  tu porównujemy m znaków (w pętli
        wypisz: znalezione wystąpienie wzorca od pozycji s+1
```

```
Naiwny(P,T)
// P - wzorzec, tablica [1..m],
// T - tekst, tablica [1..n],
  for s=0 to n-m
      if P[1..m]==T[s+1 .. s+m]   //  tu porównujemy m znaków (w pętli
        wypisz: znalezione wystąpienie wzorca od pozycji s+1
```

Stwierdzenie. Złożoność pesymistyczna naiwnego algorytmu wyszukiwania wzorca w tekście jest $\Theta(m(n-m+1))$ gdzie m to długość wzorca a n to długość tekstu.

Dowód. Wzorzec `aaa...aa` długości m, tekst `aaaaa...aaaaa` długości n.

```
aaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 aaaaaaa                   aaaaaaa
```

Algorytm Rabina-Karpa

Idea algorytmu

```
Rabin-Karp(P,T,d,q)
// P - wzorzec, tablica [1..m],
// T - tekst, tablica [1..n],
// hasz() schemat haszujący ciąg znaków długości m; wynik typu int
    p = hasz(P[1..m])
    t = hasz(T[1..m])
    // wyliczone: wartość p "kodująca" P[1..m]
    // oraz       wartość t "kodująca" T[s+1..s+m] dla s==0
    // kodowanie niejednoznaczne! (haszowanie)
    for s=0 to n-m
      // Niezmiennik: p==hasz(P[1..m]), t==hasz(T[s+1..s+m])
      if p==t // podejrzenie, że może być wystąpienie wzorca
        if P[1..m]==T[s+1 .. s+m]   //  tu porównujemy m znaków (w pęt
          wypisz: znalezione wystąpienie wzorca od pozycji s+1
      if s < n-m  // czy tekst się nie skończył?
        t =  hasz( T[s+2, ... , s+m,s+1+m])
        // wykorzystaj przy tym znany hasz(T[s+1,s+2, ... , s+m])
```

Schemat haszowania

załóżmy, że znaki występujące w tekście i wzorcu mają kody w zakresie od 0 do `d-1` dla pewnej wartości `d`

$$\texttt{hasz(P[1..m])} = \texttt{P[1]}\cdot\texttt{d}^{m-1} + \texttt{P[2]}\cdot\texttt{d}^{m-2} + \ldots + \texttt{P[m]}\cdot\texttt{d}^{0}$$

$$\texttt{hasz(T[s+1..s+m])} = \texttt{T[s+1]}\cdot\texttt{d}^{m-1} + \texttt{T[s+2]}\cdot\texttt{d}^{m-2} + \ldots + \texttt{T[s+m]}\cdot\texttt{d}^{0}$$

$$\texttt{hasz(T[s+2..s+m+1])} = \texttt{T[s+2]}\cdot\texttt{d}^{m-1} + \ldots + \texttt{T[s+m]}\cdot\texttt{d}^{1} + \texttt{T[s+m+1]}\cdot\texttt{d}^{0}$$

wtedy

$$\texttt{hasz(T[s+2..s+m+1])} = \texttt{(hasz(T[s+1..s+m])} - \texttt{T[s+1]}\cdot\texttt{d}^{m-1})\cdot\texttt{d} + \texttt{T[s+m+1]}$$

W dodatku operacje wykonujemy modulo `q` (dość duża liczba pierwsza)

Pseudokod według podręcznika z doprecyzowaniem operacji arytmetycznych modulo q i zadbaniem o nieprzekracznie zakresu liczb typu `int` .

```
Rabin-Karp(P,T,d,q)
// P - wzorzec, tablica [1..m],
// T- tekst, tablica [1..n],
// d - rozmiar alfabetu (np. 128),
// q - liczba pierwsza (np. 27077)
  h=1;
  for i=1 to m-1
    h = (h*d) % q   // wyliczy h = (d do potęgi m-1) modulo q
  p=0
  t=0
  for i=1 to m
    p = (d*p+P[i])%q
    t = (d*t+T[i])%q
  // wyliczone: wartość p "kodująca" P[1..m]
  // oraz       wartość t "kodująca" T[s+1..s+m] dla s==0
  // kodowanie niejednoznaczne! (haszowanie)
  for s=0 to n-m
    if p==t
      if P[1..m]==T[s+1 .. s+m]   // tu porównujemy m znaków (w pętli)
        wypisz: znalezione wystąpienie wzorca od pozycji s+1
    if s < n-m  // czy tekst się nie skończył?
      t1=(T[s+1]*h)%q;
      if (t<t1) t=t+q;
      t = (d*(t-t1)+T[s+m+1])%q;
      // czyli t = (d*(t-T[s+1]*h)+T[s+1+m])%q, gdzie arytmetyka jest
      // modulo q (mnożenie i odejmowanie)
      // to wylicza t "kodujące"         T[s+2, ... , s+m,s+1+m]
      // na podstawie t "kodującego" T[s+1,s+2, ... , s+m]
```

Stwierdzenie. Złożoność pesymistyczna algorytmu Rabina-Karpa jest $\Theta(m(n-m+1))$ gdzie m to długość wzorca a n to długość tekstu.

Dowód. Tak samo jak dla algorytmu naiwnego.

Stwierdzenie. Złożoność oczekiwana algorytmu Rabina-Karpa jest $\Theta(n)$ gdzie n to długość tekstu, przy założeniach, że długość wzorca jest mniejsza od długości tekstu, ilość wystąpień wzorca w tekście jest ograniczona przez stałą oraz liczba q jest większa od długości wzorca.

Dowód. Czas działania jest $\Theta(n)$ plus czas potrzebny na wykonanie sprawdzeń czy `P[1..m]==T[s+1 .. s+m]` gdy `p==t` , czyli gdy wartości haszowane dla wzorca i fragmentu tekstu `T[s+1 .. s+m]` się zgodziły. Tak się dzieje gdy rzeczywiście jest dopasowanie wzorca i takich sytuacji jest $O(1)$ lub gdy nie ma dopasowania wzorca ale `p==t` .

Wyliczając pseudolosowo t, z jednakowym prawdopodobieństwem trafiamy w każdą z wartości z zakresu od 0 do $q-1$, czyli prawdopodobieństwo, że `p==t` jest $1/q$ i takie wyliczenie jest robione $n-m+1$ razy czyli nie więcej niż n razy.

Ostatecznie, ponieważ $q > m$ i $m < n$ oczekiwany czas to

$$\Theta(n) + O(m) \cdot (O(1) + n \cdot 1/q) = \Theta(n) + O(m+n) = \Theta(n)$$

Algorytm Knutha-Morrisa-Pratta

Wyliczanie tablicy prefiksów `pi`

```
Prefix-Function(P)
// P - wzorzec, tablica [1..m]
  pi[1]=0
  k=0
  for q=2 to m
    // mamy: P[1..k]==P[...q-1], k==pi[q-1]
    while k>0 and P[k+1] != P[q]
       k=pi[k]
    if P[k+1] == P[q]
       k=k+1
    pi[q]=k
```

Przykład

```
q       123456789
pi[q]   001234512
P       abababaab
          ababab...
            abab....
              ab.....
               ab....
```

Wyliczanie tablicy prefiksów `pi`

```
Prefix-Function(P)
// P - wzorzec, tablica [1..m]
  pi[1]=0
  k=0
  for q=2 to m
    // mamy: P[1..k]==P[...q-1], k==pi[q-1]
    while k>0 and P[k+1] != P[q]
       k=pi[k]
    if P[k+1] == P[q]
       k=k+1
    pi[q]=k
```

Wyszukiwanie wzorca

```
KMP(T,P)
// T - tekst, tablica [1..n]
// P - wzorzec, tablica [1..m]
  pi = Prefix-Function(P)
  q=0
  for i=1 to n
    // mamy: P[1..q]==T[...i-1]
    while q>0 and P[q+1] != T[i]
       q=pi[q]
    if P[q+1] == T[i]
       q=q+1
    if q==m
       wypisz: znalezione wystąpienie wzorca od pozycji i-m+1
       q=pi[q]
```

Stwierdzenie. Złożoność pesymistyczna algorytmu Knutha-Morrisa-Pratta jest $\Theta(m+n)$ gdzie m to długość wzorca a n to długość tekstu.

Dowód. Funkcja `KMP`:

Jeżeli pominiemy pętle `while` to pętla `for` ma czas $\Theta(n)$.

W każdym obrocie pętli `while` zmniejszane jest `q`. Ale `q` może być zwiększone tylko n razy o 1, zatem sumaryczna ilość zmniejszeń `q` we wszystkich obrotach pętli `while`, dla wszystkich obrotów pętli `for` jest nie większa niż n. Czyli sumaruczny czas wykonania wszystkich pętli `while` jest $O(n)$

Razem więc mamy czas $\Theta(n)$ dla `KMP`.

Funkcja `Prefix-Function(P)`: rozumując analogicznie jak dla `KMP` dostajemy $\Theta(m)$.

Ostatecznie razem jest $\Theta(n+m)$

Stwierdzenie. Algorytm Knutha-Morrisa-Pratta jest poprawny, to znaczy znajduje wszystkie wystąpienia wzorca w tekście.

Dowód. Opiera się na obserwacji, że dla wzorca P i jego prefiksu P_q prefiksy

$$P_{pi[q]},\ P_{pi[pi[q]]},\ \ldots$$

są wszystkimi sufiksami właściwymi P_q