Wyszukiwanie wzorca w tekście w stałej pamięci i czasie liniowym

Łukasz Selwa

Maj 2022

1 Wprowadzenie

Naiwny algorytm wyszukiwania wzorca w w tekście t wymaga jedynie stałej pamięci ale zajmuje czas $O(|w| \cdot |t|)$. Znane są algorytmy rozwiązujące problem wyszukiwania wzorca w czasie liniowym $O(|w| + |t|)$, jak algorytm Knutha-Morrisa-Pratta, jednak najczęściej potrzebują liniowej dodatkowej pamięci.

W pracy „Time-Space-Optimal String Matching" Z. Galil i J. Seiferas jako pierwsi przedstawili algorytm do wyszukiwania wzorca działający w czasie liniowym i stałej pamięci. Niniejszy tekst jest oparty na późniejszej pracy M. Crochemore i W. Rytter „Squares, Cubes, and Time-Space Efficient StringS earching", która zawiera uproszczony opis i dowód oryginalnego algorytmu Galil-Seiferas.

2 Algorytm

2.1 Idea algorytmu

Idea algorytmu Galil-Seiferas polega na zmodyfikowaniu działania algorytmu Morris–Pratta tak aby oszczędzić trzymania w pamięci całej tablicy prefiksosufiksów. Dokonuje się tego poprzez dekompozycje wzorca w na słowa u, v, takie że $w = uv$, u jest stosunkowo krótkim słowem a v nie zawiera dużych prefiksów. W słowie t szukając wystąpień wzorca w będziemy znajdować wystąpienia v i dla każdego wystąpienia naiwnie sprawdzać czy bezpośrednio przed v występuje u. Okazuję się, że da się to zrobić tak, żeby cały algorytm dalej działał w czasie liniowym od $|t| + |w|$.

2.2 Opis Algorytmu

Definicja 2.1. ***k-HRP****. Dla $k \geqslant 3$ mówimy, że słowo v jest k-wysoce-powtarzanym-prefiksem (k-HRP, ang. k-highly-repeating-prefix) słowa w, jeśli v^i jest prefiksem w dla pewnego $i \geqslant k$ oraz v jest słowem pierwotnym. Będziemy oznaczać k-HRP przez samo HRP jeśli k będzie jasne z kontekstu.*

Zauważmy, że najkrótszy okres słowa v^2 to $|v|$. Inaczej v nie byłoby słowem pierwotnym. Z definicji k-HRP v^2 jest prefiksem w. Dzięki temu możemy zdefiniować zakres v.

Definicja 2.2. ***Zakres.*** *Niech v będzie k-HRP słowa w. Przedział $[L, R]$ nazywamy zakresem v jeśli $L = |v^2|$ oraz $[L, R]$ jest najdłuższym przedziałem takim, że dla każdego $i \in [L, R]$ najkrótszy okres $w[1...i]$ jest równy $|v|$.*

Lemat 2.1. *Niech v_1, v_2 to k-HRP słowa w, $|v_1| < |v_2|$ oraz $[L_1, R_1], [L_2, R_2]$ to zakresy odpowiednio v_1, v_2. Wtedy $R_1 < L_2$.*

Dowód. Niech z będzie prefiksem w długości R_1. Jeśli $L_2 \leqslant R_1$ to z tego wynika, że v_2^2 jest prefiksem z. Ponieważ, $|v_1| < |v_2|$ i $|v_1|$ jest okresem $z = v_2^2 z'$ to $|v_1|$ jest też okresem v_2^2. Z lematu o okresowości wynika, że $\gcd(|v_1|, |v_2|)$ też musi być okresem v_2^2. Ale $\gcd(|v_1|, |v_2|) < |v_2|$ stąd sprzeczność z pierwotnością v_2. □

Dla słowa x, przez $p(x)$ oznaczamy najkrótszy okres x.

Lemat 2.2. *Niech $v_1, ..., v_r$ będzie sekwencją wszystkich k-HRP słowa w. Niech $[L_i, R_i]$ to zakres v_i. Wtedy dla każdego niepustego prefiksu u słowa w zachodzi:*

$$\begin{array}{ll} p(u) = L_i/2 & \textit{jeśli } |u| \in [L_i, R_i] \textit{ dla pewnego } i \\ p(u) > |u|/k & \textit{w przeciwnym przypadku} \end{array}$$

Dowód. Jeśli $|u|$ nie należy do żadnego przedziału $[L_i, R_i]$ to każdy okres u musi być większy od $|u|/k$. Jeśli $|u| \in [L_i, R_i]$ to u musi być prefiksem v_i^e, dla $e \geqslant 2$. Z pierwotności v_i najkrótszy okres u to $|v_i| = L_i/2$. □

Załóżmy, że w jest wzorcem, dla którego v_1 jest jedynym k-HRP z zakresem $[L_1, R_1]$. Wtedy możemy znaleźć wszystkie wystąpienia w w tekście t w czasie liniowym. Niech $m = |w|, n = |t|$.

Algorithm 1 SIMPLE-TEXT-SEARCH

```
pos ← 0, j ← 0
while pos + m ⩽ n do
  while j < m ∧ w[j + 1] = t[pos + j + 1] do
    j ← j + 1
  if j = m then
    return Match at position pos.
  if j ∈ [L_1, R_1] then
    pos ← pos + L_1/2
    j ← j − L_1/2
  else
    pos ← pos + ⌊j/k⌋ + 1
    j ← 0
```

Poprawność algorytmu wynika z lematu 2.2. Algorytm działa w czasie liniowym ponieważ cały czas rośnie wartość $k \cdot pos + j$. Łatwo też zauważyć, że dodatkowa pamięć wymagana przez algorytm jest stała.

Definicja 2.3. ***k-perfekcyjna-dekompozycja.*** *Dla danego w mówimy, że słowa u, v są k-perfekcyjną dekompozycją jeśli $w = uv$, v ma co najwyżej jedno k-HRP oraz $|u| < 2p(v)$.*

Zauważmy, że jeśli dla wzorca w mamy k-perfekcyjną dekompozycję to możemy łatwo znaleźć wszystkie wystąpienia wzorca w czasie liniowym i stałej pamięci za pomocą algorytmu 1. W czasie liniowym możemy wyszukać wszystkie wystąpienia v i dla każdego sprawdzić czy jest poprzedzane przez u. Złożoność zamortyzuje się do czasu liniowego ponieważ $|u| \leqslant 2|v|$. Przedstawimy algorytm znajdujący k-perfekcyjną dekompozycję.

2.3 Znajdowanie dekompozycji

Ustalmy $k \geqslant 3$. Przez $\mathrm{HRP}\, i(x)$ oznaczamy i-ty najkrótszy HRP słowa x.

Lemat 2.3. *Załóżmy, że x ma* HRP*, niech $z = \mathrm{HRP}(x)$ oraz $x = zx'$.*

1. *Jeśli istnieje* $\mathrm{HRP}\, 2(x)$ *to* $|\mathrm{HRP}\, 2(x)| > 2 \cdot |\mathrm{HRP}\, 1(x)|$
2. *Jeśli istnieje* $\mathrm{HRP}\, 1(x')$ *to* $\mathrm{HRP}\, 1(x)$ *jest prefiksem* $\mathrm{HRP}\, 1(x')$

Dowód. Załóżmy nie wprost, że $|\mathrm{HRP}\, 2(x)| \leqslant 2 \cdot |\mathrm{HRP}\, 1(x)|$. Ponieważ z definicji $\mathrm{HRP}\, 1(x)^2$ jest prefiksem x to $\mathrm{HRP}\, 2(x)$ jest prefiksem $\mathrm{HRP}\, 1(x)^2$. Co przeczy pierwotności $\mathrm{HRP}\, 2(x)$.

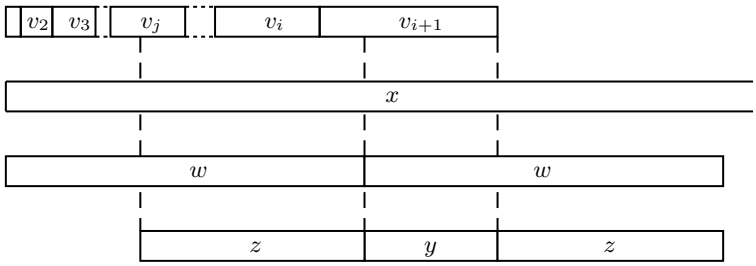
Ponieważ $k \geqslant 3$ to $(\mathrm{HRP}\, 1(x))^2$ jest prefiksem x'. Jeśli zachodziłoby $|\mathrm{HRP}\, 1(x')| < |\mathrm{HRP}\, 1(x)|$ to albo $\mathrm{HRP}\, 1(x)$ nie byłoby słowem pierwotnym albo nie byłoby najkrótszym HRP słowa x. □

Definicja 2.4. ***Ciąg czynników.*** *Dla słowa x ciąg $V(x) = (v_1, v_2, ...)$ definiujemy następująco, $v_1 = \mathrm{HRP}\, 1(x)$. Niech x' będzie słowem takim, że $x = v_1x'$. Wtedy $x_2 = \mathrm{HRP}(x')$ i tak dalej dopóki* HRP *istnieją.*

Z lematu 2.3 wynika, że ciąg czynników jest rosnący ze względu na długość.

Lemat 2.4 (Kluczowy Lemat)**.** *Niech $V(x) = (v_1, v_2, v_3...)$ będzie ciągiem czynników i niech $\mathrm{HRP}\, 2(x)$ istnieje. Niech i będzie największą liczbą taką, że $|v_1...v_i| < |\mathrm{HRP}\, 2(x)|$. Wtedy jeśli v_{i+1} istnieje to $|v_{i+1}| \geqslant |\mathrm{HRP}\, 2(x)|$*

Szkic dowodu. Niech $w = \mathrm{HRP}\, 2(x)$. Załóżmy nie wprost, że $|v_{i+1}| \leqslant |w|$. Zauważmy, że v_{i+1} musi przecinać dwa wystąpienia w, to jest istnieją słowa u, y takie, że $v = uy$, $y \neq \epsilon$ jest prefiksem w a u jest sufiksem w. Niech z spełnia $w = yz$. Ponieważ $v_1...v_i$ jest prefiksem w, a w jest prefiksem $v_1...v_{i+1}$ to dla pewnego j, z jest podsłowem $v_j...v_{i+1}$. Ponieważ $|v_{i+1}| < |w|$ to $j < i + 1$. Przykładowa sytuacja pokazana jest na rysunku 1.



Rysunek 1: Ilustracja do dowodu lematu.

Należy rozważyć dwa przypadki. Pierwszy kiedy v_j jest prefiksem z. Wtedy za pomocą lematu 2.3 można pokazać, że w nie jest słowem pierwotnym. Drugi przypadek jest kiedy początek zy znajduje się ściśle wewnątrz v_j. Za pomocą lematu 2.3 można pokazać, że w tym przypadku v_j nie może być słowem pierwotnym.

Definicja 2.5. ***Pozycja specjalna.*** *Lemat 2.4 pozwala na zdefiniowanie pojęcia pozycji specjalnych. Niech $V(x) = (v_1, ...)$ to ciąg czynników x. Pierwszą pozycją specjalną słowa x jest długość słowa $v_1...v_i$ takiego, że v_{i+1} albo nie istnieje albo $|v_1...v_{i+1}| \geqslant \mathrm{HRP}\,2(x)$. Jeśli $x = v_1...v_i x'$ to kolejną pozycją specjalną definiujemy analogicznie dla x'. Robimy tak dopóki dla x istnieje* $\mathrm{HRP}\,2(x)$.

Twierdzenie 2.1 (O dekompozycji)**.** *Niech j będzie ostatnią specjalną pozycją słowa x. Niech $u = x[1...j], v = x[j+1...n]$. Wtedy uv jest k-perfekcyjną dekompozycją x.*

Z definicji v nie może mieć $\mathrm{HRP}\,2(v)$ pozostało zauważyć, że $|u| < p(v)$. Kluczową obserwacją jest fakt, że dla kolejnych pozycji specjalnych j, j' zachodzi $2 \cdot |\,\mathrm{HRP}\,2(x[j...])| \leqslant |\,\mathrm{HRP}\,2(x[j'...])|$. Wynika to z lematów 2.3. 2.4. Na podstawie twierdzenia 2.1 możemy skonstruować algorytm do wyznaczania k-perfekcyjnej dekompozycji.

Algorithm 2 PERFECT-DECOMPOSITION

```
j ← 0, hrp1 ← |HRP 1(x)|, hrp2 ← |HRP 2(x)|
while hrp1 ≠ NULL ∧ hrp2 ≠ NULL do
    j ← j + hrp1
    hrp1 ← |HRP 1(x[j...n])
    if hrp1 ⩾ hrp2 then
        hrp2 ← |HRP 2(x[j...n])
```

Galil Seiferas pokazali, że jeśli potrafimy obliczyć $\mathrm{HRP}\,1(x)$ oraz $\mathrm{HRP}\,2(x)$ w czasie proporcjonalnym do $|\,\mathrm{HRP}\,2(x)|$ to powyższy algorytm jest liniowy. Dowód tego faktu pominiemy. Pokażemy za to sam algorytm wyznaczania $\mathrm{HRP}\,1(x)$ i $\mathrm{HRP}\,2(x)$ w odpowiednim czasie.

2.4 Wyznaczanie k-HRP

Przedstawimy jak wyznaczyć wszystkie HRP słowa w o długości n. Załóżmy, że mamy policzony ciąg pierwszych HRP $v_1, v_2, ..., v_r$ i ich odpowiednie zakresy $[L_1, R_1], [L_2, R_2], ..., [L_r, R_r]$. W tedy z lematu 2.1 wiemy, że kolejny HRP może pojawić się od pozycji $R_r/2$.

Ustalmy pozycję $pos > R_r/2$. Załóżmy, że sprawdziliśmy wszystkie pozycje w przedziale $[R_r/2, pos)$ i wiemy, że żadna z nich nie jest HRP. Niech j będzie liczbą taką, że $w[1...j]$ to najdłuższy wspólny prefiks w oraz $w[pos...n]$. Jeśli $pos \cdot k \leqslant pos + j$ to $v[1...pos]$ jest k-HRP słowa w, ponieważ $(w[1...pos])^k$ jest prefiksem w oraz w jest słowem pierwotnym bo inaczej znaleźlibyśmy k-HRP wcześniej. Mamy też od razu obliczony zakres v_{r+1}, ponieważ z definicji $L_{r+1} = 2 \cdot pos$ oraz $R_{r+1} = pos + j$. Jeśli natomiast $pos \cdot k > pos + j$ to $w[1...pos]$ nie może być k-HRP.

Zastanówmy się gdzie może być następny potencjalny koniec k-HRP. Jeśli j nie należy do żadnego zakresu $[L_i, R_i]$ to z lematu 2.2 najkrótszy okres $w[1...j]$ jest większy od j/k. W takim przypadku możemy więc przesunąć pozycję pos o $\lfloor j/k \rfloor + 1$ i mieć pewność, że nie pominęliśmy kolejnego k-HRP.

Jeśli natomiast $j \in [L_i, R_i]$ dla pewnego k-HRP v_i to z lematu 2.2 wiemy, że $p(w[1...j]) = L_i/2$. Stąd następna potencjalna pozycja jest równa $pos + L_i/2$. Ale ponieważ $j \geqslant L_i$ to wiemy, że najdłuższy prefiks w i $w[pos...n]$ ma co najmniej długość $j - L_i/2$. Taka analiza prowadzi do następującego algorytmu.

Algorithm 3 PREPROCESS

```
spcopes ← empty list
pos ← 1, j ← 0
while pos + m < n do
  while pos + j < m ∧ w[j + 1] = w[pos + j + 1] do
    j ← j + 1
  if k · pos ⩽ pos + j then
    Append [2 · pos, pos + j] to list spcopes.
  if j ∈ [L_i, R_i] for some [L_i, R_i] in scopes then
    pos ← pos + L_i/2
    j ← j − L_i/2
  else
    pos ← pos + ⌊j/k⌋ + 1
    j ← 0
return scopes
```

Podobnie jak w przypadku algorytmu 1 podany algorytm jest liniowy ponieważ cały czas rośnie wartość $k \cdot pos + j$.

Zauważmy, że jeśli zatrzymamy algorytm gdy znajdziemy HRP 1 i HRP 2 to dostaniemy algorytm obliczający pierwsze dwa HRP w stałej pamięci i liniowy od długości HRP 2. To kończy opis działania algorytmu Galil-Seiferas w wersji Crochemore-Rytter.