Algorytm Turbo_BM

Adrian Siwiec

June 20, 2020

Opis Algorytmu.

Algorytm Turbo_BM jest modyfikacją algorytmu Boyera-Moore'a, która zużywa tylko stałą dodatkową pamięć oraz przyspiesza złożoność pesymistyczną do $2n$. Nie będziemy wykonywać żadnego dodatkowego preprocessingu – tak jak w oryginalnym algorytmie korzystać będziemy z tablicy BM obliczonej dla wzorca.

Sam pomysł algorytmu Turbo_BM jest bardzo prosty – podczas skanowania będziemy zapisywać jedno podsłowo wzorca ($memory$), o którym wiemy że pasuje do tekstu na obecnej pozycji. Dzięki temu możemy pominąć ten fragment przy sprawdzaniu dopasowania, oraz w przypadku jego braku możemy wykonać tzw. *Turbo-shift*.

Opiszmy sytuację, w której możemy wykonać Turbo-shift. Niech x będzie najdłuższym suffixem wzorca, który pasuje do tekstu na danej pozycji, a będzie pierwszą literą wzorca która nie pasuje, a niech y ($memory$) będzie poprzednio zapamiętanym podsłowem, które również pasuje do wzorca (patrz fig. 1). Załóżmy również, że x i y są rozłączne i y jest dłuższy od x. Po wykonaniu Turbo-shift zapominamy o y (patrz kod *boyer_moore_turbo*), więc w poprzednim kroku wykonaliśmy zwykłe przesunięcie (zgodnie z tablicą BM), znane z algorytmu Boyera-Moore'a. W tej sytuacji ax jest suffixem y, oraz litery a i b tekstu są o siebie oddalone o $|y|$. Ale suffix $y \dots x$ ma okres długości $|y|$ (z definicji BM). Możemy więc przesunąć wzorzec o $|y| - |x|$ do przodu i to właśnie przesunięcie nazywać będziemy *Turbo-shift*.

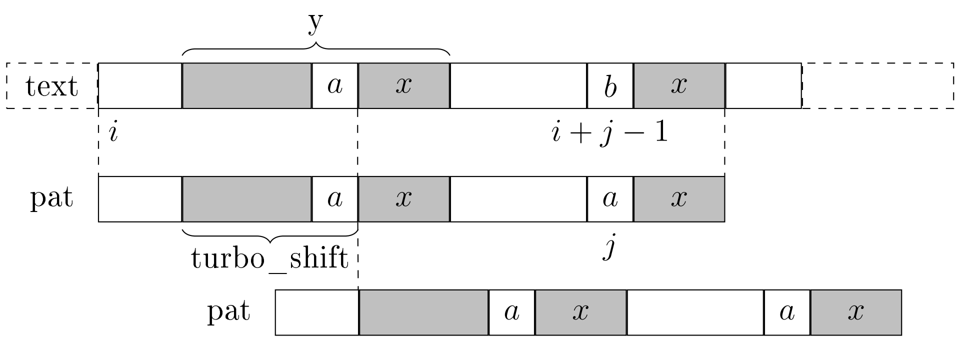

Figure 1: Turbo-shift

Analiza złożoności.

Analiza złożoności algorytmu Turbo_BM jest dużo prostsza niż analiza algorytmu Boyera Moore'a bez żadnych modyfikacji.

Twierdzenie. *Algorytm Turbo_BM wykonuje co najwyżej* $2n$ *porównań.*

Proof. Podzielimy wyszukiwanie wzorca na etapy, każdy etap będzie się składał z dwóch operacji: skanowania i przesuwania wzorca. Podczas etapu k niech Suf_k oznacza suffix wzorca, który pasuje do tekstu, a $shift_k$ niech oznacza długość o którą przesuniemy wzorzec podczas etapu k.

Przsunięcie na etapie k nazwiemy *krótkim*, gdy $2 * shift_k < |Suf_k| + 1$. Wyróżnimy 3 typy etapów:

(1) Etap, po którym następuje przeskok przy skanowaniu dzięki *memory*.

(2) (1) nie zachodzi i wykonujemy długie przesunięcie.

(3) (1) nie zachodzi i wykonujemy krótkie przesunięcie.

Zastosujemy analizę kosztu zamortyzowanego. Zdefiniujmy $cost_k$: dla etapu typu (1) $cost_k = 1$ – będzie to tylko koszt porównania litery, która się nie zgadza. Resztę kosztu przenosimy na pozostałe typy etapów. Dla etapów (2) i (3) $cost_k = |Suf_k| + 1$. Wystarczy nam pokazać, że $\sum_k cost_k < 2 * \sum_k shift_k$. To nam wystarczy, bo $\sum_k shift_k \leq |t|$.

Dla etapu k typu (1) $cost_k = 1$ jest trywialnie mniejszy od $2 * shift_k$. Dla etapu k typu (2) $cost_k = |Suf_k| + 1 \leq 2 * shift_k$ z definicji długiego przesunięcia.

Wystarczy rozważyć etapy typu (3). Jedyna możliwość wykonania krótkiego przesunięcia zachodzi, gdy nie wykonujemy Turbo-shiftu. Ustawiamy wtedy zmienną *memory*, co prowadzi do potencjalnego Turbo-shiftu na etapie $k+1$. Rozważmy dwa przypadki etapu (3):

(a) $|Suf_k|+shift_k \leq |pat|$. Wtedy z definicji Turbo-shifta mamy: $|Suf_k| - |Suf_{k+1}| \leq shift_{k+1}$, a więc:

$$cost_k = |Suf_k| + 1 \leq |Suf_{k+1}| + shift_{k+1} + 1 \leq shift_k + shift_{k+1}.$$

(b) $|Suf_k| + shift_k > |pat|$. Wtedy mamy: $|Suf_{k+1}| + shift_k + shift_{k+1} \geq |pat|$, oraz:

$$cost_k \leq |pat| \leq 2 * shift_k - 1 + shift_{k+1}.$$

Możemy założyć, że na etapie $k+1$ zachodzi przypadek (b), bo daje on gorsze ograniczenie na $cost_k$. Jeśli etap $k+1$ jest typu (1), mamy: $cost_k + cost_{k+1} \leq 2 * shift_k + shift_{k+1}$. Gdy na etapie $k+1$ zachodzi nierówność $|Suf_{k+1}| \leq shift_{k+1}$ wtedy mamy: $cost_k + cost_{k+1} \leq 2 * shift_k + 2 * shift_{k+1}$.

Wystarczy rozważyć sytuację, gdy na etapie $k+1$ mamy $|Suf_{k+1}| > shift_{k+1}$. To oznacza, że na etapie $k+1$ wykonujemy standardowe przesunięcie (a nie Turbo-shift). Wtedy aplikujemy indukcyjnie nasze powyższe rozumowanie do etapu $k+1$, a ponieważ wtedy może zajść tylko przypadek (a) otrzymujemy: $cost_{k+1} \leq shift_{k+1} + shift_{k+2}$, a więc: $cost_k + cost_{k+1} \leq 2 * shift_k + 2 * shift_{k+1} + shift_{k+2}$.

Musimy jeszcze tylko domknąć indukcję: jeśli na wszystkich etapach i od k do $k+j$ mamy $|Suf_i| > shift_i$, wtedy: $cost_k + \ldots + cost_{k+j} \leq 2 * shift_k + \ldots + 2 * shift_{k+j} + shift_{k+j+1}$.

Niech k' będzie pierwszym etapem po etapie k na którym $|Suf_{k'}| \leq shift_{k'}$. Wtedy otrzymujemy $cost_k + \ldots + cost_{k'} \leq 2 * shift_k + \ldots + 2 * shift_{k'}$ co kończy dowód. $\square$