Zadanie 1 z listy 5 - "Kompresja Danych"

Łukasz Klasiński

3 maja 2020

Zadanie 1

Mamy słowo $w_k = a^{k(k+1)/2}(ba^k)^{(k+1)^2}$.

Zobaczmy najpierw jak wygląda konstrukcja gramatyki dla w_k, która ma wielkość $O(log_k)$. Najpierw mamy produkcję słowa startowego: $S \to LR$ Gdzie L będzie prawymi a, natomiast R prawymi.

Jak wygląda tworzenie słowa złożonego z a^k elementów: Rozkładamy wpierw k na postać binarną $k = 2^{i_1} + 2^{i_2} + 2^{i_3} \dots$ Wtedy produkcja a^k wygląda następująco:

$$A_K \to A_{i_1} A_{i_2} A_{i_3} \dots$$

Natomiast kolejne produkcje A_j tworzymy następująco:

$$A_j \to A_{j/2} A_{j/2}$$

$$A_1 \to a$$

Otrzymujemy wtedy $\lceil logk \rceil = O(logk)$ produkcji.

Teraz chcemy znaleźć produkcję tworzącą słowo składające się z $(k+1)/2$ elementów x. Robimy to tak samo jak wcześniej - szukamy rozkładu binarnego i robimy dodatkowe produkcje:

$$A_{\frac{K+1}{2}} \to A_{\frac{K+1}{2}_i} A_{\frac{K+1}{2}_{i+1}} \dots$$
$$A_{\frac{K+1}{2}_1} \to x$$

Możemy teraz stworzyć produkcję konstruującą słowo $a^{k(k+1)/2}$ podstawiając pod x z poprzedniej produkcji A_K

$$L \to A_{\frac{K+1}{2}}$$

Mamy już zatem lewe wyrażenie szukanego słowa. Zostało stworzenie produkcji dla $(ba^k)^{(k+1)^2}$. Zauważmy, że

$$(k+1)^2 = k^2 + 2k + 1$$

Zatem wystarczy znaleźć produkcje dające kolejne długości tej sumy. Jedynkę mamy za darmo bo $(ba^k)^1 = bA_K$ $(ba^k)^{2k} = 2 \times prod(ba^k)^k$ natomiast $k^2 = k * k = k \times prod(ba^k)^k$ Wystarczy zatem zrobić produkcję, które aplikuje elementy k razy oraz produkcję $(ba^k)^k$ Najpierw $(ba^k)^k$ - podobnie jak wcześniej korzystamy z rozkładu k i robimy produkcję

$$BA_K \to BA_{i_1} BA_{i_2} \dots$$

$$BA_j \to BA_{j/2} BA_{j/2}$$
$$BA_1 \to bA_K$$

Oraz identycznie produkcję która zaaplikuje produkcję BA_K - k razy:

$$K_K \to K_K i_1 K_K i_2 \dots$$

$$K_1 \to BA_K$$

Ostatecznie mamy produkcję dla prawej strony:

$$R \to bA_K BA_K BA_K K_K$$

Wystarczy teraz zlepić całość w

$$S \to A_{\frac{K+1}{2}}$$

Jaką mamy wielkość gramatyki - 4 * O(logk) = O(logk)

Dlaczego to jest optymalna wielkość gramatyki? Wiemy z wykładu, że gramatyka konstruowana przez algorytm LZ77 jest większa od optymalnej gramatyki o jakąś stałą $\leq logn$ gdzie $n = |w|$. Algorytm na początku dopasuje długi ciąg złożony z samych a i zapisze to jako jedna krotka. Następnie zmachuje ba^k jednokrotnie, następnie dwukrotnie, czterokrotnie - łącznie $log(k+1)^2 = 2log(k+1)$ razy ostatecznie mamy zatem kod długości $2log(k+1) * log(|w|) = O(logk) * log(|w|)$. Zatem $O(logk)$ musi być optimum (na mocy tw. z wykładu).

Jak wygląda kodowanie LZ78 - zakładamy nieograniczony słownik. Wpierw zobaczmy L. Na początku wstawi do pustego słownika a i wypluje $(0, a)$. W następnym kroku zmatchuje a i doda do słownika aa (wypluje (1, a)). I tak dalej - łącznie zamieni a^1 na krotkę, a^2 na krotkę ... a^k na krotkę, ponieważ $\sum_1^k = k(k+1)/2$.

Prawa część będzie wyglądać następująco - algorytm zobaczy b na początku wyrazu, zatem nie znajdzie tego w słowniku i doda to jako nowa krotka. Następnie zobaczy że a^k jest w słowniku i doda do słownika $a^k b$. W kolejnej iteracji dodamy $a^k ba$. Potem $a^{k-1}b$. Ogólnie otrzymamy w słowniku wszystkie możliwe kombinacje tego gdzie możemy wstawić b pomiędzy (lub na końcach) a^k oraz wszystkie możliwe kombinacje $a^i ba^j$, ba^i gdzie $i, j = 0..k$. Zatem ustawiamy i, j na k możliwych sposobów mamy k^2 sposobów na to. Zatem na wyjściu otrzymamy co najmniej $O(k^2)$ krotek. Ostatecznie mamy $O(k) + O(k^2) \geq \Omega(K)$