Aproksymacja najkrótszego nadsłowa

Mateusz Tokarz

July 4, 2020

Opis Problemu

Problem najkrótszego nadsłowa w wersji decyzyjnej jest problemem NP zupełnym. W związku z tym pojawiało się wiele podejść do aproksymacji optymalizacyjnej wersji problemu. Tarhio i Ukkonen [1] oraz Turner [2] napisali dwie niezależne prace naukowe, w których opisano algorytmy aproksymacyjne dla rozważanego problemu. Mimo, że autorzy twierdzili, że skonstruowane przez nich algorytmy powinny być 2-aproksymacyjne, to nie udało im się udowodnić żadnego ograniczenia.

W tym dokumencie przeanalizujemy algorytm opisany przez Minga Li, który jest pierwszą aproksymacją, dla której udało się udowodnić współczynnik - $\log(n)$.

Algorytm

Zakładamy, że podane na wejściu słowa spełniają warunek, iż żadne nie jest podsłowem innego. Możemy tak zrobić, ponieważ wyeliminowanie tego typu sytuacji jest niezmiennicze ze względu na wynik i zajmuje wielomianowy czas.

Definicja. *Niech S będzie pewnym skończonym zbiorem słów nad alfabetem $\mathcal{A}$, w którym żadne słowo nie jest podsłowem innego. Zdefiniujmy następujący zbiór:*

$$R_S = \{(s, s', l) \in S^2 \times \mathbb{N} : \ l < min(|s|, |s'|) \wedge s[(|s|-l+1) \dots |s|] = s'[1 \dots l]\}.$$

Funkcje scalająca

$$m : R_s \to \mathcal{A}^*,$$

to odwzorowanie spełniające

$$m(s, s', l) = ss'[(l+1)\dots|s'|].$$

Definiujemy także zbiór

$$M(S) = \overrightarrow{m}(R_S).$$

Lemat. *Dla danych słów $s_1, s_2 \in S$, zbiór R_S zawiera conajwyżej*

$$2 \cdot min(|s_1|, |s_2|)$$

elementów typu

$$(s_1, s_2, l_1) \text{ lub } (s_2, s_1, l_2),$$

gdzie l_1 i l_2 to dowolne liczby naturalne.

Dowód powyższego lematu wynika bezpośrednio z definicji zbioru R_S. Naturalny wniosek z powyższego lematu, to skończoność zbiorów R_S i $M(S)$.

Algorytm

1. Input to $S = \{s_1, \dots, s_n\}$. Jeżeli $|S| = 1$, to zwróć s_1.
2. Zdefiniujmy $T = \emptyset$.
3. Niech s będzie słowem minimalizującym

   $$\min_{s \in M(S)} \frac{|s|}{v(s)},$$

   gdzie

   $$v(s) = \sum_{a \in A(s)} |a|$$

   dla $A(s)$ będącego zbiorem podsłów s z S. Wrzucamy s do T, natomiast wszystkie słowa z $A(s)$ wyrzucamy z S.
4. Jeżeli $|S| \leq 1$, to $S = T \cup S$ i przechodzimy do 1.
5. W przeciwnym przypadku przechodzimy do 3.

Złożoność jest oczywiście wielomianowa - przy każdym powrocie do punktu 1, moc zbioru S zmniejszyła się przynajmniej dwukrotnie, natomiast sumaryczna długość słów znajdujących się w nim może się tylko zmniejszyć.

Wyszukiwanie minimum i modyfikacja zbiorów w punktach 3 i 4 jest trywialnie realizowane w czasie wielomianowym.

Pozostaje ograniczyć współczynnik aproksymacji algorytmu.

Współczynnik aproksymacji

Twierdzenie. *Algorytm przedstawiony powyżej jest* $\log(n)$ *aproksymacyjny.*

Dowód. Niech $s_1, \ldots, s_m$ będzie ułożeniem słów wejściowych w kolejności ich występowania w optymalnym rozwiązaniu - jeżeli słowo występuje kilka razy, to rozważamy najbardziej lewe wystąpienie. Dzielimy słowa na następujące grupy:
Grupa G_1 zawiera $s_1, \ldots, s_i$, gdzie s_i to ostatnie słowo takie, że pierwsze wystąpienie s_1 nachodzi na pierwsze wystąpienie s_i w optymalnym rozwiązaniu. Grupa G_2 budowana jest w analogiczny sposób, z tym że zaczynamy konstrukcje od s_{i+1}. W ten sposób dzielimy całe wejście $s_1, \ldots, s_m$ na rozłączne grupy $G_1, \ldots, G_k$.

Dla każdej grupy G_i wyróżniamy elementy b_i, t_i - pierwsze i ostatnie słowa (we wcześniej wspomnianym porządku) budujące grupę. Jako że b_i nachodzi na t_i, to istnieje takie l_i, że wszystkie słowa z grupy G_i są podsłowami $m(b_i, t_i, l_i)$. Zauważmy, że

$$\sum_{i=1}^{k} |m_i(b_i, t_i, l_i)| \leq 2n,$$

gdzie n to długość optymalnego rozwiązania, ponieważ każda litera słowa optymalnego jest pokrywana przez maksymalnie dwie grupy.

Wprowadźmy teraz pojęcie kosztu słowa w kontekście analizowanego przez nas algorytmu. Załóżmy, że słowo w odpadało ze zbioru S poprzez bycie podsłowem s.

$$cost(w) = \frac{|s|}{v(s)} \cdot |w|.$$

Zauważmy, że koszt całego algorytmu (długość słowa wynikowego) jest równa nie więcej niż długość wszystkich słów, które powstały w trakcie

pierwszej fazy opróżniania zbioru S - elementów $s_1, \ldots, s_m$. Niech $S_h = A(m(B_h, T_h, L_h))$ to zbiór elementów wyrzuconych h-tym kroku wspomnianej fazy.

$$COST \leq \sum_{h=1}^{r} |m(B_h, T_h, L_h)| = \sum_{i=h}^{r} \frac{|m(B_h, T_h, L_h)|}{v(m(B_h, T_h, L_h))} \cdot v(m(B_h, T_h, L_h)) =$$

$$\sum_{h=1}^{r} \sum_{s \in S_h} \frac{|m(B_h, T_h, L_h)|}{v(m(B_h, T_h, L_h))} \cdot |s| = \sum_{h=1}^{r} \sum_{s \in S_h} cost(s) = \sum_{s \in S} cost(s).$$

□

Rozważmy teraz grupę G_j i koszt, jaki płacimy za wyrzucenie jej elementów. Grupę G_j dokładnie przed fazą h będziemy oznaczać G_j^h. Sumę długości słów w zbiorze A oznaczamy $||A||$.

$$Cost(G_j) = \sum_{h=1}^{k} \sum_{s \in (G_j^{h+1} \setminus G_j^h)} cost(s) = \sum_{h=1}^{k} \frac{|m(B_h, T_h, L_h))|}{v(m(B_h, T_h, L_h))} \cdot (||G_j^{h+1}|| - ||G_j^h||).$$

Jako że nasz algorytm wybierał scalenia $m(B_h, T_h, L_h)$ realizujące minimalne wartości, to

$$\sum_{h=1}^{k} \frac{|m(B_h, T_h, L_h))|}{v(m(B_h, T_h, L_h))} \cdot (||G_j^h|| - ||G_j^{h+1}||) \leq \sum_{h=1}^{k} \frac{|m(b_j, t_j, l_j))|}{v(m(b_j, t_j, l_j))} \cdot (||G_j^h|| - ||G_j^{h+1}||) =$$

$$\sum_{h=1}^{k} \frac{|m(b_j, t_j, l_j))|}{||G_j^h||} \cdot (||G_j^h|| - ||G_j^{h+1}||) \leq |m(b_j, t_j, l_j)| \cdot Harm(||G_j||),$$

gdzie

$$Harm(n) = \sum_{i=1}^{n} \frac{1}{i}.$$

Ostatnia nierownosc wynika natychmiast z następującego lematu:

Lemat. *Niech $a_1, \ldots, a_s$ będzie ciągiem liczb naturalnych spełniających*

$$a_1 < a_2 < \cdots < a_s.$$

Wtedy

$$\sum_{i=1}^{s-1} \frac{a_i - a_{i+1}}{a_i} \leq Harm(a_1).$$

Dowód. Zauważmy, że

$$\frac{a_i - a_{i+1}}{a_i} = \underbrace{\frac{1}{a_i} + \frac{1}{a_i} + \cdots + \frac{1}{a_i}}_{a_i - a_{i+1}} \leq \sum_{j=0}^{a_i - a_{i+1} - 1} \frac{1}{a_i - j},$$

zatem

$$\sum_{i=1}^{s-1} \frac{a_i - a_{i+1}}{a_i} \leq \sum_{i=1}^{s-1} \sum_{j=0}^{a_i - a_{i+1} - 1} \frac{1}{a_i - j} =$$

$$\sum_{i=1}^{s-1} \frac{1}{a_i} + \cdots + \frac{1}{a_{i+1} - 1} \leq Harm(a_1).$$

□

Jako że n jest co najwyżej wielomianowo większe niż każde $||G_j||$, to dostajemy oszacowanie

$$Cost(G_j) \leq |m(b_j, t_j, l_j))| \cdot \log(n),$$

co z poprzednim lematem skutkuje w

$$COST \leq 2n \log(n).$$

Twierdzenie. *Istnieje rodzina wejść, dla której algorytm jest dokładnie* $\log(n)$ *aproksymacją.*

Dowód. Zdefiniujmy ciąg zbiorów $S_0, \ldots, S_{k-1}$ w następujący sposób:

$$S_i = \{a^j b^{4^k - j} : \ j \in \{\frac{4^{k-i}}{2}, \frac{4^{k-i}}{2} + 1, \ldots, 4^{k-i}\}\}$$

oraz zbiór

$$V = \{cb^{i-1}ca^{4^i - i - 1} : \ i \in \{1, \ldots, k-1\}\}.$$

Wejście do naszego algorytmu to suma wszystkich S_i oraz V.
Zauważmy, że

$$|cb^{i-1}ca^{4^i - i - 1}| = 1 + i - 1 + 1 + 4^i - i - 1 = 4^i,$$

zatem konkatenacja v wszystkich słów w V ma długość

$$4 + 4^2 + \ldots 4^{k-1} = \frac{4}{3}(4^{k-1} - 1).$$

Słowo $va^{4^k}b^{4^k}$ jest nadsłowem całego inputu i jego długość jest ograniczona przez $3 \cdot 4^k$.
Prosta analiza pokazuje, że nasz algorytm wrzuci w pierwszej fazie do zbioru T kolejno słowa

$$cb^{k-1}ca^{4^k-1}b^{4^k-\frac{4^k}{2}},$$

$$cb^{k-2}ca^{4^{k-1}-1}b^{4^k-\frac{4^{k-1}}{2}},$$

$$\dots$$

$$cbca^3b^{4^k-2}.$$

Jak widać najmniejsze możliwe nadsłowo to ich konkatencja, której długość jest rzędu $k \cdot 4^k$ co kończy dowód. □

Bibliografia

[1] J. Tarhio and E. Ukkonen. A Greedy approximation algorithm for constructing shortest common superstrings. *Theoretical Computer Science 57, 131-145, 1988.*

[2] J. Turner. Approximation algorithms for the shortest common superstring problem. *Information and Computation 83, 1-20, 1989.*