Faster suffix sorting

Algorytm obliczania tablicy sufiksowej autorstwa N. Larssona i K. Sadakane [Sad]

Paweł Palenica

Wstęp

Algorytmy konstrukcji tablicy sufiksowej są jednym z klasycznych zagadnień algorytmów tekstowych. Podczas gdy znane są algorytmy tworzące tę strukturę w czasie liniowym od długości słowa wejściowego, często wymagają one dużego narzutu pamięciowego. Zastosowaniem tablicy sufiksowej dla którego duży narzut pamięciowy wynikający z użycia np. drzewa sufiksowego może być niepożądany jest transformata Burrowsa-Wheelera - odwracalna transformacja tekstu która zmienia kolejność znaków w tekście aby zwiększyć ciągi podobnych znaków, co może pomóc w kompresji.

Problem który będzie rozwiązywany definiujemy następująco. Niech $X = x_1x_2\dots x_n\$$ będzie ciągiem znaków złożonym z ciągu wejściowego długości n z dopisanym symbolem \$, który jest leksykograficznie przed wszystkimi symbolami ciągu wejściowego. Jako $S_i,\ 1 \leq i \leq n+1$ będziemy oznaczać sufiks X zaczynający się w pozycji i. Wyjściem algorytmu ma być tablica indeksów I (*tablica sufiksowa*) taka że $S_{I[k-1]}$ jest leksykograficznie mniejsze od $S_{I[k]}$ dla $0 < k \leq n$.

Algorytm tworzenia tablicy sufiksowej zaprezentowany przez N. Larssona i K. Sadakane działa w czasie $O(n\ log\ n)$ gdzie n jest długością wejściowego tekstu. Autorzy rozbudowują podejście [Udi] i korzystają z techniki ‘podwajania‘ znanej z algorytmu Karpa, Millera, Rosenbera która polega na użyciu pozycji sufiksów po fazie sortowania jako kluczy do sortowania dwukrotnie dłuższych sufiksów w kolejnej fazie. Autorzy formalizują to podejście definiując pojęcie *h-order*, czyli porządek leksykograficzny sufiksów rozważając jedynie h początkowych znaków każdego sufiksu. Zayważmy, że *h-order* niekoniecznie jest jednoznaczny jeśli $h < n$.

Obserwacja. *Sortowanie sufiksów przy użyciu klucza w postaci pary złożonej z pozycji sufiksu S_i w h-order oraz pozycji sufiksu S_{i+h} w h-order daje sortowanie sufiksów w 2h-order.*

Definicja. *Dla tablicy sufiksów I która jest w h-order definiujemy*

- *Maksymalna (względem długości) spójna sekwencja sufiksów w I które mają te same początkowe h znaków to* ***grupa***
- *Grupa składająca się z co najmniej 2 sufiksów jest* ***grupą nieposortowaną***
- *Grupa składająca się z jednego sufiksu jest* ***grupą posortowaną***
- *Maksymalna spójna sekwencja posortowanych grup jest* ***wspólną grupą posortowaną***

Grupy będziemy numerować żeby móc przeprowadzać obliczenia na poszczególnych grupach z osobna. Dla grupy $I[f \dots g]$ numer tej grupy to g. Ponadto tablica $V[i] = g$ zawiera informację, że sufiks S_{i+1} znajduje się w grupie o numerze g. Dodatkowo będziemy korzystać z tablicy L która trzyma długości nieposortowanych grup oraz długości wspólnych grup posortowanych. Dla rozróżnienia, długości tych późniejszych będą trzymane jako liczba ujemna. Tablicę L użyjemy w podstawowej wersji algorytmu. Udoskonalenia omówione pod koniec tego omówienia pozwolą nam obyć się bez osobnej tablicy na długości grup.

Obserwacja. *Gdy I jest w h-order, każdy sufiks w* ***wspólnej grupie posortowanej*** *jest unikalnie rozróżnialny od wszystkich innych sufiksów przy pomocy pierwszych h symboli.*

Mając na uwadze powyższy niezmiennik, będziemy ‘przeskakiwać‘ nad grupami posortowanymi.

Podstawowy algorytm będzie się prezentował następująco:

1. Umieść sufiksy $1 \ldots n+1$ w tablicy I. Posortuj I używając x_i jako klucza dla sufiksu i (pierwsza litera). Ustaw $h = 1$
2. Dla każdego sufiksu S_{i+1} ustaw numer grupy do której należy w $V[i]$
3. Dla każdej nieposortowanej grupy lub wspólnej grupy posortowanej zapisz jej długość (lub zanegowaną długość) do tablicy L
4. Przesortuj każdą nieposortowaną grupę z osobna za pomocą *ternary quick sorta* (czyli takiego co dzieli na podtablice $[< pivot, = pivot, > pivot]$), używając $V[S_k + h - 1]$ jako klucza dla każdego sufiksu S_k w grupie
5. Zaznacz pozycje podziału pomiędzy nierównymi kluczami w dla każdej przetworzonej nieposortowanej grupy
6. Podwój h. Stwórz nowe grupy poprzez podział na zaznaczonych pozycjach (aktualizując odpowiednio V i L)
7. Zakończ jeśli I składa się tylko z jednej posortowanej grupy. W przeciwnym przypadku idź do kroku 4.

Na pierwszy rzut oka można by wyciągnąć wniosek, że algorytm działa w czasie $O(n\ log^2\ n)$. Dokładniejsza analiza pozwala na ograniczenie przez $O(n\ log\ n)$. Na potrzeby analizy zakładamy, że *ternary quicksort* dzieli zawsze w medianie (możliwe do uzyskania przy wydajnym algorytmie znajdowania mediany, nieefektywne w praktyce). Proces sortowania całej tablicy sufiksowej (nie tylko jednego quicksorta) przedstawimy jako ternarne drzewo obliczeń.

Lemat. *Długość ścieżki od korzenia do liścia jest ograniczona z góry przez* $2log\ n + 3$

Proof. Każdy węzeł 'środkowy' na ścieżce odpowiada podwojeniu *h-order* w którym znajduje się rozważana tablica. Takich węzłów może być więc $log\ n + 1$ Z faktu, że używamy mediany po zejściu lewym lub prawym dzieckiem długość rozważanej tablicy zmniejsza się o połowę. Znów takich węzłów będzie maksymalnie $log\ n + 1$ □

Na danym poziomie drzewa ilość wykonanej pracy będzie $O(n)$ bo to obliczenie odpowiada wykonaniu splita na rozłączne podtablice. Stąd dostajemy złożoność $O(n\ log\ n)$ bo aktualizacja podtablic odpowiedzialnych za grupy również jest wykonywana w czasie liniowym od rozmiaru podtablicy.

W algorytmie autorzy wprowadzają usprawnienia które nie wpływają one negatywnie na czas wykonania. Opiszemy tutaj większość z nich (te które zostały wykorzystane w implementacji).

Eliminacja tablicy długości L. Tablica długości dla grup nieposortowanych jest nam zbędna ze względu na to jak numerujemy grupy. Przypomnijmy, numer grupy $I[f \ldots g]$ to g, czyli $V[f-1] = g$ a w konsekwencji rozmiar grupy to $V[f-1] - g + 1$. Do spamiętania rozmiarów wspólnych grup posortowanych możemy użyć tablicy I. Wpólne grupy posortowane nie pojawią się w wywołaniu quicksorta a tylko tam korzystamy z I w trakcie działania algorytmu. Jedynym problemem pozostaje odzyskanie porządku w I na samym końcu algorytmu, ale tutaj wystarczy, że skorzystamy z tablicy V trzymającej numery grup (indywidualne grupy posortowane mają różne numery), więc $I[V[i]] = i$ dla $0 \leq i < n+1$ odzyska tablicę sufiksową na końcu. Ostatecznie otrzymujemy następującą procedurę odzyskania długości grupy: jeśli $I[i] < 0$ to $I[i \ldots i - I[i] - 1]$ jest wspólną grupą posortowaną. W przeciwnym wypadku $I[i \ldots i + V[I[i] - 1]]$ jest nieposortowaną grupą.

Połączenie sortowania i aktualizowania tablic. Zauważmy, że wykonanie połączenia dwóch następujących wspólnych grup posortowanych możemy wykonywać dopiero przy kroku 4 algorytmu gdy iterujemy się po kolejnych nieposortowanych grupach w I przeskakując nad wspólnymi grupami posortowanymi. Aktualizacja tablic dla grup nieposortowanych w czasie wykonania sortowania musi być wykonana z uwagą na to żeby nie zmienić wartości w V które będą wykorzystane jako klucze sortowania. Fakt wykorzystania quicksorta zapewnia nam, że ustalenie kolejności aktualizacji tablic jest proste:

1. Podziel tablice względem *pivot* na trzy partycje
2. Wywołaj się rekurencyjnie na sekcji '$<$ *pivot*'
3. Zaktualizuj numery grup w sekcji '$=$ *pivot*' która się staje w całości nową grupą
4. Wywołaj się rekurencyjnie na sekcji '$>$ *pivot*'

Bibliografia

[Sad] N. Jesper Larsson Kunihiko Sadakane. *Faster Suffix Sorting*. URL: `http://www.larsson.dogma.net/ssrev-tr.pdf`. (accessed: 18.06.2020).

[Udi] Gene Mayers Udi Manber. *Suffix Arrays: A New Method for On-Line String Searches*. URL: `https://courses.cs.washington.edu/courses/cse590q/00au/papers/manber-myers_soda90.pdf`. (accessed: 18.06.2020).