

LCS w czasie oczekiwanym $O(ND)$

Wojciech Grabis

June 2020

1 Wprowadzenie

Algorytm omówiony w pracy służy do poszukiwania najdłuższego wspólnego podciągu. Powstał w oparciu o pracę **An $O(ND)$ Difference Algorithm and Its Variations** autorstwa **Eugene W. Myers**.

2 Definicje

Zanim przejdziemy do algorytmu zdefiniujemy graf edycji, definicje wspólnego podciągu oraz pewne pomocnicze określenia.

2.1 Graf edycji

Mając dane słowa $A = a_1a_2 \dots a_N$ oraz $B = b_1b_2 \dots b_M$ jako graf edycji G oznaczamy skierowany graf, dla którego zbiór wierzchołków to punkty siatki (x, y) , $x \in [0, N]$ i $y \in [0, M]$, natomiast na krawędzi składają się:

- $(x - 1, y) \rightarrow (x, y)$ dla każdego $x \in [1, N]$ oraz $y \in [0, M]$.
- $(x, y - 1) \rightarrow (x, y)$ dla każdego $x \in [0, N]$ oraz $y \in [1, M]$.
- $(x - 1, y - 1) \rightarrow (x, y)$ jeśli $a_x = b_y$, taka krawędź definiuje pewne dopasowanie pomiędzy wzorcami, punkty dla których $a_x = b_y$ nazwiemy punktami dopasowania.

Definicja (Podciąg słowa). *Podciąg słowa A to każde słowo powstałe poprzez usunięcie 0 lub więcej symboli z A . Natomiast wspólny podciąg dwóch słów A i B to podciąg każdego z nich.*

2.2 Graf edycji, a wspólny podciąg

Definicja (Ślad). *Śladem długości L nazywamy sekwencję L punktów dopasowania $(x_1, y_1), (x_2, y_2) \dots (x_L, y_L)$, gdzie dla każdego $1 \leq i < L$: $x_i < x_{i+1} \wedge y_i < y_{i+1}$. Jest to pewna sekwencja diagonalnych krawędzi. Ślad możemy powiązać z pewną ścieżką pomiędzy $(0, 0)$ a (N, M) .*

Każdy ślad grafu edycji definiuje pewien wspólny podciąg słów A i B, który powstaje poprzez odpowiednie symbole zdefiniowane przez punkty dopasowania.

Definicja (Odległość edycyjna). *Odległość edycyjna D to ilość operacji edycji które przeprowadzamy na tekście A żeby przeprowadzić go w tekst B. Jeśli spojrzymy na ścieżkę która zawiera pewien ślad L to każda krawędź pozioma w tej ścieżce definiuje pewną operację usunięcia symbolu z tekstu A, natomiast krawędzi pionowe definiują operację dodania symboli do tekstu A za odpowiednim indeksem w ciągu A. Możemy zauważyć że wartość odległości edycyjnej jest powiązana z długością śladu L , tj $D = N + M - 2L$.*

Idea algorytmu będzie opierała się na wyszukiwaniu najdłuższego śladu pomiędzy punktami $(0,0)$ oraz (N,M) , czyli znalezieniu ścieżki która zawiera największą ilość krawędzi diagonalnych, powyższy ślad wyznaczy nam najdłuższy wspólny podciąg słów A i B.

Pozostały nam jeszcze dwie definicje, które będą wykorzystywane w algorytmie:

Definicja (D-ścieżka). *Jako D-ścieżkę definiujemy ścieżkę w grafie edycji rozpoczynającą się w punkcie $(0,0)$, posiadającą dokładnie D krawędzi nie-diagonalnych.*

Definicja (k-diagonala). *W celu wprowadzenia algorytmu zdefiniujemy numerację diagonal. K-diagonala, to taka krawędź diagonalna, która składa się z punktów (x, y) , takich że $x - y = k$. Zgodnie z tym krawędzi są numerowane od $-M$ do N .*

Na podstawie definicji k-diagonali zauważmy, że krawędź pionowa (pozioma) której punkt startowy leży na k-diagonali, posiada drugi koniec krawędzi na $(k-1)$ -diagonali (odpowiednio $(k+1)$ -diagonali dla poziomej). Zbiór diagonal znajdujących się na końcu krawędzi pionowej/poziomej nazywamy ogonem (w przypadku gdy koniec krawędzi nie jest początkiem żadnej diagonal, wtedy ogon nazywamy pustym).

3 Algorytm

3.1 Poszukiwanie najdłuższej ścieżki

Algorytm będzie opierał się na wyszukiwaniu najdalszych D-ścieżek w k-diagonali. Przed wprowadzeniem pseudokodu, najpierw omówimy pewne właściwości związane z diagonalami oraz ścieżkami

Lemat. *Każda D-ścieżka musi kończyć się na k-diagonali, takiej że $k \in \{-D, -D+2, \dots, D-2, D\}$*

Dowód. Dowód indukcyjny. Dla $D = 0$ zauważmy, że 0-ścieżka może kończyć się tylko na diagonalu 0, ponieważ oba indeksy (x, y) na tej ścieżce muszą być sobie równe (nie możemy wykorzystać żadnej krawędzi poziomej lub pionowej z definicji 0-ścieżki). Następnie zauważmy, że D-ścieżka składa się z pewnej (D-1)-ścieżki, pewnej poziomej/pionowej krawędzi oraz ogona. Z założenia indukcji mamy że ścieżka D-1 kończy się na pewnej k krawędzi gdzie $k \in \{-D+1, -D+3, \dots, D-3, D-1\}$, natomiast nowy ogon będzie składał się z pewnej krawędzi $k \pm 1$ (bazując na obserwacji z k-diagonalami), stąd zbiór diagonali na których kończy się D-ścieżka to $k_D \in \{-D+1 \pm 1, -D+3 \pm 1, \dots, D-3 \pm 1, D-1 \pm 1\}$, czyli mamy $k_D \in \{-D, -D+2, \dots, D-2, D\}$. \square

Definicja (Najdalej sięgająca D-ścieżka). *D-ścieżkę nazywamy najdalej sięgającą w diagonalu k, jeśli jest to ścieżka której końcowy punkt ma największy numer kolumny (odpowiednio numer wiersza z definicji k-diagonali) spośród wszystkich D-ścieżek w k diagonalu.*

Lemat (Rozbicie najdalej sięgającej D-ścieżki). *Najdalej sięgająca D-ścieżka w diagonalu k może być rozbita na najdalej sięgającą (D-1)-ścieżkę kończącą się na diagonalu k - 1, krawędź poziomą oraz najdłuższy ogon z tej krawędzi, lub w najdalej sięgającą (D-1)-ścieżkę w diagonalu k + 1, krawędź pionową oraz najdłuższy ogon z tej krawędzi.*

Dowód. Wybranie odpowiedniej najdalszej (D-1)-ścieżki w diagonalu k+1/k-1 związane jest ze wcześniej wspomnianą obserwacją na temat ogona krawędzi poziomej/pionowej. Natomiast w celu udowodnienia faktu że to najdalej sięgająca (D-1)-ścieżka założymy że (D-1)-ścieżka składająca się na kompozycję najdalej sięgającej D-ścieżki nie jest najdalej sięgająca w diagonalu k, ale możemy zaobserwować że skoro D-ścieżka musi sięgać dalej niż (D-1)-ścieżka, to oznacza że do ogona tej D-ścieżki będziemy mogli się też dostać

z końca ogona najdalszej (D-1)-ścieżki przy pomocy odpowiedniej krawędzi niediagonalnej, więc zawsze możemy najdalszą D-ścieżkę rozłożyć na najdalszą (D-1)-ścieżkę. \square

Na bazie obu lematów wprowadzamy algorytm obliczania punktu końca najdalej sięgającej D-ścieżki w diagonalu k. Zauważmy że odległość edycyjna $D \in [0, M + N]$, więc będziemy kolejno badali możliwe odległości edycyjne, aż dojdziemy do punktu (N, M) .

Algorithm Obliczanie odległości edycyjnej

```

V : [0] + [0] * 2(m + n)
V[1]  $\leftarrow$  0
for D  $\leftarrow$  0 to M + N do
  for k  $\leftarrow$  -D to D in steps of 2 do
    if k = -D or k  $\neq$  D and V[k - 1] < V[k + 1] then
      x  $\leftarrow$  V[k + 1]
    else
      x  $\leftarrow$  V[k - 1] + 1
    end if
    y  $\leftarrow$  x - k
    while x < N and y < M and  $a_{x+1} = b_{y+1}$  do
      V[k]  $\leftarrow$  x
    end while
    if x  $\geq$  N and y  $\geq$  M then
      return D
    end if
  end for
end for

```

Powyższy algorytm pozwala nam w czasie $O((M + N)D)$ wyznaczyć odległość edycyjną (a co za tym idzie również długość najdłuższego wspólnego podciągu) przy wykorzystaniu $O(D)$ pamięci.

Niestety problem pojawia się przy wyznaczeniu symboli na które składa się najdłuższy wspólny podciąg. Musielibyśmy po każdej iteracji zapisywać tablicę V, w celu wyznaczenia z których punktów przechodziliśmy w tej ścieżce, co sprawiałoby że musielibyśmy wykorzystać $O(D^2)$ pamięci.

W tym celu w następnej części omówimy algorytm bazujący na przeszukiwaniu najdalszych ścieżek, który będzie działał w $O(D)$ pamięci.

3.2 Algorytm w liniowej pamięci

Finalny algorytm na poszukiwanie najdłuższego wspólnego podciągu będzie opierał się na strategii dziel i rządź. Algorytm będzie korzystał z poszukiwania najdalej sięgającej ścieżki z punktu $(0, 0)$ oraz odwrotnej ścieżki z punktu (N, M) . W celu obliczenia odwrotnej ścieżki będziemy korzystać z poprzedniego algorytmu, z tym że tym razem będziemy szukali ścieżki z (N, M) w kierunku $(0, 0)$ odwracając krawędzie z grafu edycji oraz minimalizując wartość Y w przeciwieństwie do maksymalizowania X z poprzedniego algorytmu (natomiast ścieżka w przód liczona jest analogicznie do poprzedniego algorytmu). Przed zaprezentowaniem algorytmu udowodnimy następujący lemat:

Lemat. *Podział D-ścieżek* *Pomiędzy punktami $(0, 0)$ i (N, M) istnieje D-ścieżka wtedy i tylko wtedy gdy istnieje $\lceil D/2 \rceil$ -ścieżka z $(0, 0)$ do punktu (x, y) oraz $\lfloor D/2 \rfloor$ -ścieżka z punktu (u, v) do (M, N) , taka że*

- $u + v \geq \lceil D/2 \rceil$ oraz $x + y \leq N + M - \lfloor D/2 \rfloor$
- $x - y = u - v$ oraz $x \geq u$.

Generalnie powyższy lemat opiera się na podziale D-ścieżki na pewne 2 ścieżki, jedna ścieżka z $(0, 0)$ a druga ścieżka, którą możemy traktować jako odwrotną ścieżkę z (N, M) w kierunku $(0, 0)$. Zauważmy że jak rozłożymy D-ścieżkę na 2 takie przeciwne ścieżki, to ogon znajdujący się na końcu ścieżki do przodu będzie pokrywał się z ogonem znajdującym się na końcu ścieżki przeciwnej, będziemy właśnie z tej właściwości korzystać szukając ścieżkę do przodu i ścieżkę od końca, aż nasze ścieżki pokryją się w pewnej diagonalu. Właśnie te diagonale z ogona które się pokrywają wyznaczają nam symbole które będą "środkiem" naszego najdłuższego wspólnego podciągu.

Następnie wykorzystamy strategię dziel i rządź rozdzielimy sobie słowa na podstawie środkowego ogona i wykonamy się rekurencyjnie na naszych podsłowach, największy wspólny podciąg to rekurencyjne wywołanie lewej części słów, znaków z ogona oraz wywołania na prawej części słów.

Algorytm będzie opierał się na odpowiedniej adaptacji poprzedniego algorytmu do przeszukiwania wpród D-ścieżek oraz wstecznych D-ścieżek, w

przypadku gdy dla jakiejś diagonali k ogony ścieżek się pokrywają zwracamy odpowiednią odległość edycyjną oraz ogon który znaleźliśmy. W przypadku gdy dla jakiegoś D , D -ścieżka wprzód pokrywa się z $(D-1)$ -ścieżką przeciwną, to zwracamy $2D - 1$, natomiast gdy D -ścieżka przeciwna pokrywa się z D -ścieżką wprzód zwracamy $2D$.

Nasz finalny algorytm na znalezienie największego wspólnego podciągu prezentuję się następująco:

Algorithm LCS(A,B,N,M)

if $N > 0$ and $M > 0$ **then**

Uruchamiamy naszą procedurę szukania środkowego ogona
 $(X, Y) \rightarrow (U, V)$ to nasz środkowy ogon, D - odległość edycyjna

if $D > 1$ **then**

$left = LCS(A[1 \dots x], B[1 \dots y], x, y)$

$middle = A[x + 1 \dots u]$

$right = LCS(A[u + 1 \dots N], B[v + 1 \dots M], N - u, M - v)$

return $left + middle + right$

else if $M > N$ **then**

return $A[1 \dots N]$

else

return $B[1 \dots M]$

end if

end if

3.3 Złożoność

Oznaczmy jako $T(P, D)$ czas działania algorytmu gdzie $P = N + M$. Nasz czas działania opisuje następująco zależność

$$T(P, D) \leq \begin{cases} \alpha PD + T(P_1, \lceil D/2 \rceil) + T(P_2, \lfloor D/2 \rfloor) & \text{if } D > 1 \\ \beta P & \text{if } D \leq 1 \end{cases}$$

gdzie $P_1 + P_2 \leq P$. Korzystając z zależności $\lceil D/2 \rceil \leq \frac{2D}{3}$ dla $D \geq 2$, możemy udowodnić że $T(P, D) \leq 3\alpha PD + \beta P$, co dowodzi że algorytm wykonuje się w $O((M + N)D)$ czasie.

Natomiast nasza złożoność pamięciowa, zauważmy, że procedura poszukiwania środkowego ogona zajmuje $O(D)$ miejsca, ze względu na wykorzystanie dwóch wektorów V , dodatkowo procedura poszukiwania środkowego ogona działa niezależnie od rekurencji, więc na całą rekurencję potrzebujemy miejsca tylko na 2 wektory, natomiast rekurencja składa się z $O(\log D)$ poziomów więc zajmuje $O(\log D)$ pamięci, stąd mamy złożoność $O(m + n)$ pamięciową.