Zadanie 8 z listy 1

Antoni Kościelski

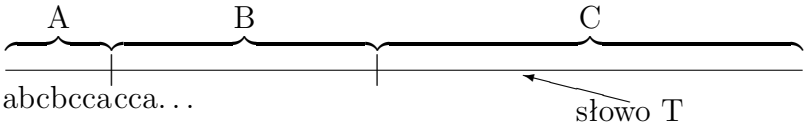
Zadanie 8 z listy 1 dotyczy problemu algorytmicznego "podaj liczbę wystąpień słowa S w tekście T". Sformułowanie tego problemu nie jest (nie było) w pełni jasne.

1 Pojęcie słowa

Aby zdefiniować informatyczne pojęcie słowa, musimy mieć alfabet. Formalnie jest to dowolny zbiór skończony. O jego elementach myślimy jak o znakach. Może to być zbiór liter, którymi się posługujemy, zbiór znaków możliwych do wprowadzenia z klawiatury komputera, albo na przykład zbiór cyfr. W językach programowania mamy zwykle typy znakowe (na przykład `char`) i pewien zbiór wartości takiego typu pełniący rolę alfabetu. Dalej będziemy zakładać, że posługujemy się ustalonym alfabetem.

Słowa to skończone ciągi znaków. Słowo jest określone, jeżeli znamy jego długość i kolejne znaki. Jeżeli a, b i c są znakami, to możemy z nich utworzyć ciąg długości 7 o wyrazach a, b, c, b, c, c oraz a. Taki ciąg często przedstawiamy jako *abcbcca* (w językach programowania zwykle ujmujemy ten napis dodatkowo w apostrofy). Programiści słowa nazywają również napisami, łańcuchami lub stringami. Słowa bywają implemntowane jako tablice, bywa że indeksowane od 0. Będziemy naśladować związaną z tym symbolikę. Jeżeli opisane wyżej słowo oznaczymy literą A, to wzory $A[0]$ i $A[3]$ będą oznaczać odpowiednio pierwszy (czyli o indeksie 0) i czwarty wyraz podanego ciągu, czyli $A[0] = a$ i $A[3] = b$. Długość słowa A będziemy oznaczać symbolem $\mid A \mid$, mamy więc $\mid A \mid = 7$.

Rysunek 1 przedstawia słowo T (jest ono reprezentowane za pomocą długiej linii). Pod tą linią zostało wymienione kilka liter, które mogłyby być początkowymi literami słowa T. Pierwsze 7 z tych liter to litery słowa A.



Rysunek 1.

Słowo T zostało dodatkowo podzielone na trzy części. Każdą z tych części można utworzyć w ten sam sposób: przez wykreślenie pewnej liczby początkowych liter słowa T, być może równej 0, oraz końcowych, także być może równej 0. Słowa, które można w ten sposób otrzymać ze słowa T nazywamy podsłowami T. Tak więc słowa A, B i C z rysunku 1 są podsłowami T.

Z drugiej strony, słowo T można otrzymać przez złączenie słów A, B i C. Złączenie jest podstawową operacją na słowach: mając dwa słowa A i B o długościach odpowiednio m i n możemy je złączyć (połączyć, skonkatenować) tworząc słowo o długości $m + n$, składa się z m liter słowa A, wymienionych w tej samej kolejności, w jakiej występują w słowie A, po

których znajduje się n liter słowa B, wymienionych w kolejności znanej ze słowa B. W językach programowania konkatenację często oznacza się symbolem $+$. Tak więc związek T ze słowami A, B i C można wyrazić wzorem $T = A+B+C$. Jednak raczej będziemy stosować inną symbolikę i wzór ten zapiszemy w postaci $T = ABC$, a także w postaci $T = abcbccaBC$ utożsamiając pojedyńcze litery z jednoliterowymi słowami.

2 Rodzaje podsłów i ich wystąpienia

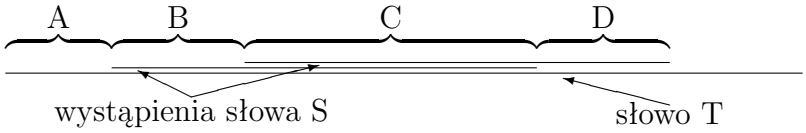
Słowo A z Rysunku 1 jest prefiksem T. Tak nazywamy niepuste słowo A o długości $< | T |$, którego kolejne litery są takie, jak litery słowa T, a wiec takie słowa A, które spełniają równości $A[i] = T[i]$ dla wszystkich $i < | A |$. Słowo T ma jeszcze dwa niewłaściwe prefiksy: słowo puste, długości 0, niezawierające żadnych liter, oraz słowo T.

Niepuste słowo C jest sufiksem słowa T, jeżeli jest krótsze niż T oraz dla wszystkich $i < |C|$ zachodzą równości $C[i] = T[| T | - | C | + i]$. Każde słowo ma też dwa niewłaściwe sufiksy, słowo puste i siebie. Słowo C przedstawione na Rysunku 1 jest sufiksem słowa T.

Z kolei wystąpieniem podsłowa B w słowie T nazywamy podsłowo B tego słowa wraz z informacją, jak B jest położone w słowie T. Można na przykład powiedzieć, że mamy na myśli wystąpienie słowa B poprzedzone prefiksem A. Sam prefiks A nie jest istotny. Można też określić wystąpienie podsłowa B podając, że jest poprzedzone prefiksem o pewnej długości k. Kolejne sposoby to stwierdzenia, że chodzi nam o podsłowo B, którego pierwsza – ewentualnie ostatnia – litera jest k-tą literą słowa T.

Zgodnie z bardzo formalną definicją wystąpieniem słowa B w słowie T nazywamy liczbę naturalną $k \leqslant | T | - | B |$ taką, że $B[i] = T[k + i]$ dla wszystkich $i < | B |$.

Dwa wystąpienia pewnego słowa są widoczne na Rysunku 2 (litery A, B i C oznaczają teraz coś innego). Oprócz linii reprezentującej słowo T są tam narysowane krótsze linie reprezentujące dwa wystąpienie słowa S. Taką sytuację łatwo stworzyć: możemy wziąc na przykład słowa $T = ababcabcababc$ i $S = abcab$.



Rysunek 2.

Takie dwa wystąpienia S rozbijają słowo T na podsłowa A, B, C, D i resztę. Pierwsze wystąpienie S jest poprzedzone prefiksem A, drugie – prefiksem AB.

Bardzo interesujące dla nas jest jeszcze słowo C z Rysunku 2. Słowo to jest prefiksem słowa S, a właściwie jego drugiego wystąpienia, i jednocześnie jest sufiksem słowa S (pierwszego wystąpienia tego słowa). Aby wygodnie mówić o takiej sytuacji, wprowadzimy pojęcie presufiksu. Niepuste słowo C będziemy nazywać presufiksem słowa S, jeżeli $| C | < | S |$ i jeżeli dla wszystkich $i < | C |$ zachodzą równości $C[i] = S[i] = S[| S | - | C | + i]$.

3 Algorytm naturalny

Najprostszy algorytm sprawdza dla kolejnych liter słowa T, czy litera pierwszą pewnego wystąpienie słowa S i zlicza takie litery. Będziemy zakładać[1], że taki algorytm dostaje jako dane

[1]Znane w pełni powinno być słowo S. Słowo T możemy czytać znak po znaku. Dobrze jest znać długość T, ale nie jest to konieczne. Wystarczy, że potrafimy poznać, że całe słowo T zostało już przeczytane.

dwa słowa T i S, reprezentowane jako tablice znaków indeksowane od 0, a także ich długości $|\,T\,|$ i $|\,S\,|$. Można zapisać go w następującej postaci:

$l \leftarrow 0$; //zmienna l będzie licznikiem wystąpień S

$i \leftarrow 0$; //zmienna i pamięta indeks pierwszej litery badanego, potencjalnego wystąpienia S

dopóki $i \leqslant |\,T\,| - |\,S\,|$ **wykonuj**

jeżeli od i-tej litery T rozpoczyna się wystąpienie S, **to** $l \leftarrow l + 1$;

$i \leftarrow i + 1$;

wypisz l.

Aby ten program był zrozumiały, trzeba wyjaśnić, jak sprawdzić, czy pewne wystąpienie słowa S w T zaczyna się od i-tej litery, lub inaczej, zgodnie z formalną definicją, czy i jest wystąpieniem S. Sprawdza to następująca procedura, która jako dane otrzymuje słowa S i T, ich długości $|\,S\,|$ i $|\,T\,|$, oraz liczbę i, dostatecznie małą:

$j \leftarrow 0$; //indeks porównywanej litery słowa S

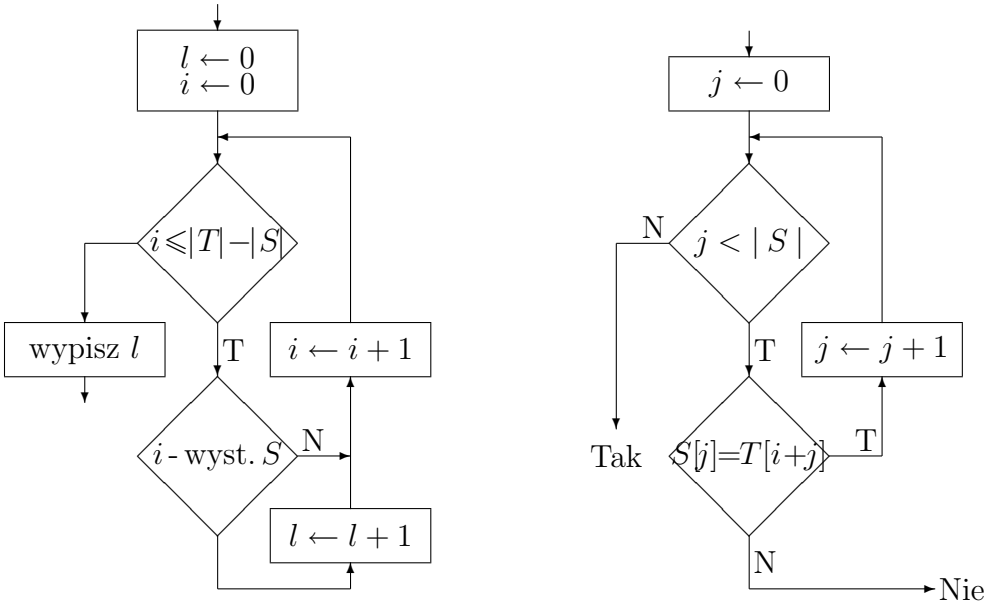
dopóki $j < |\,S\,|$ oraz **wykonuj**

jeżeli $S[j] \neq T[i + j]$, **to** zwróć, że i nie jest wystąpieniem S;

$j := j + 1$

zwróć, że i jest wystąpieniem S.

Dwuczęściowy schemat blokowy powyższego algorytmu znajduje się na rysunku 3.



Rysunek 3.
Schematy blokowe algorytmu naturalnego (po lewej) i procedury sprawdzającej, czy i jest wystąpieniem S (po prawej).

Nietrudno zauważyć, że złożoność algorytmu naturalnego jest proporcjonalna do $|\,T\,| \cdot |\,S\,|$.

4 Prostsze algorytmy

Wydaje się, że w wielu przypadkach istnieją prostsze algorytmy rozwiązujące analizowany problem. Tak jest na przykład, gdy liczymy wystąpienia słowa ujętego w nawiasy (słowo takie zaczyna się nawiasem otwierającym, kończy – zamykającym, wewnątrz tego słowa nie występuje nawias otwierający), albo gdy liczymy wystąpienia słowa w sensie gramatycznym, a więc, gdy słowo S jest ograniczone spacjami a wewnątrz niego spacja nie występuje, a nawet, gdy słowo S zaczyna się literą, która nie występuje w nim na dalszych pozycjach.

Przykładem takiego algorytmu może być następujący:

$l \leftarrow 0$; //zmienna l będzie licznikiem wystąpień S

$j \leftarrow 0$; //zmienna j przebiega indeksy liter słowa S

dopóki nie zostało przeczytane całe słowo T **wykonuj**

wczytaj do t kolejną literę słowa T;

jeżeli $t = S[j]$, **to** $j \leftarrow j + 1$;

jeżeli $j = |\, S \,|$, **to** $l \leftarrow l + 1$; $j \leftarrow 0$

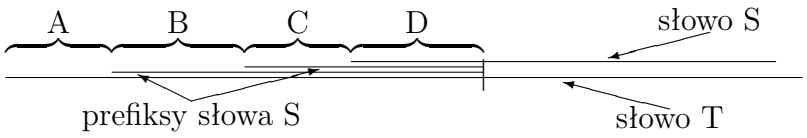
w przeciwnym razie $j \leftarrow 0$

wypisz l.

Podany algorytm działa w czasie proporcjonalnym do $|\, T \,|$, ale nie jest w pełni poprawny, wymaga, aby słowo S spełniało pewne założenia, na przykład wyżej wymienione.

5 Idea kolejnego algorytmu

Przypuśćmy, że czytamy od lewej do prawej słowo T szukając w nim wystąpień słowa S. Być może udało nam się stwierdzić w słowie T pewien prefiks słowa S. Taka sytuacja jest przedstawiona na Rysunku 4, przeczytanym prefiksem S może być słowo BCD. Kolejna litera słowa T mogła okazać się różna od kolejnej litery słowa S



Rysunek 4.

W tej sytuacji wiemy, że prefiksu BCD nie można przedłużyć do wystąpienia S, ale wiemy też, że właśnie przeczytaliśmy prefiks BCD, a kolejne wystąpienie S powinno zaczynać się na prawo od końca A. Może nim być wystąpienie S poprzedzone prefiksem ABC. Zauważmy, że wtedy słowo D jest presufiksem słowa BCD.

Gdybyśmy wcześniej zbadali presufiksy BCD, moglibyśmy znać długość D. Wtedy sprawdzanie, czy po prefiksie ABC znajduje się wystąpienie S moglibyśmy zacząć nie od litery słowa T o indeksie $|ABC|$, a od litery o indeksie $|ABCD|$, czyli od miejsca, do którego słowo T zostało już przeczytane. Mogłoby to pozwolić na znalezienie wszytkich wystąpień S po jednorazowym przeczytaniu słowa T.

6 Zadanie pomocnicze: wylicz presufiksy S

6.1 Własności presufiksów

Kilka własności presufiksów jest łatwe do zauważenia. Na przykład mamy

Lemat 6.1 *Słowo X jest presufiksem A wtedy i tylko wtedy, gdy X jest najdłuższym presufiksem A, lub gdy jest presufiksem najdłuższego presufiksu A.* □

Lemat ten mówi dwie rzeczy. Po pierwsze, bardzo ważne są najdłuższe presufiksy: jeżeli je znamy, to w pewnym sensie znamy wszystkie inne. Po drugie, jeżeli znamy najdłuższe presufiksy, to ten lemat podaje precyzyjną, rekurencyjną definicję zbioru wszystkich presufiksów.

Presufiksy długości > 1 można też wyliczać rekurencyjnie w oparciu o następujący

Lemat 6.2 *Słowo $X \neq a$ jest presufiksem Aa wtedy i tylko wtedy, gdy jest postaci $X = Ya$ dla pewnego Y będącego presufiksem słowa A (a jest tu pewnym znakiem, Aa oznacza słowo A z dopisaną na końcu literą a, czyli konkatenację A i jednoliterowego słowa a).* □

6.2 Sformułowanie problemu

Będziemy rozwiązywać następujące zadanie:

Dane: niepuste słowo S, czyli tablica znaków indeksowana liczbami od 0 do $|\,S\,|-1$,

Wynik: tablica liczbowa F indeksowana od 1 do $|\,S\,|$, w której $F[i]$ jest długością najdłuższego presufiksu słowa $S[0]S[1]\ldots S[i-1]$, czyli prefiksu długości i słowa S (w szczególności $F[i]=0$, jeżeli $S[0]S[1]\ldots S[i-1]$ ma tylko presufiksy niewłaściwe, wtedy 0 jest długością najdłuższego presufiksu krótszego niż całe słowo, ale niewłaściwego).

6.3 Algorytm rozwiązujący powyższy problem

Dane: niepuste słowo S,

$i \leftarrow 1$; //będziemy porównywać pewien prefiks słowa S z pewnym sufiksem, i to indeks wskazujący literę sufiksu.

$F[1] \leftarrow 0$; //słowo jednoliterowe $S[0]$ nie ma właściwych presufiksów. Stąd obliczenia zaczynamy od ustalenia, czy jednoliterowy sufiks $S[1]$ jest presufiksem słowa $S[0]S[1]$ (jest prefiksem $S[0]$) i początkowa wartość i jest równa 1.

$j \leftarrow 0$; //j to indeks wskazujący literę sufiksu, początkowo wskazuje pierwszą literę S. Zawsze będzie zachodzić nierówność $j < i$.

dopóki $i <|\,S\,|$ **wykonaj**

//w tym momencie słowo $A = S[0]S[1]\ldots S[j-1]$ jest najdłuższym, jeszcze nieprzeanalizowanym presufiksem słowa $B = S[0]S[1]\ldots S[i-1]$ oraz są już określone wartości $F[1], F[2], \ldots, F[i]$.

jeżeli $S[i] = S[j]$, //z lematu 6.2: $AS[j]$ jest presufiksem $BS[i]$

to $F[i+1] \leftarrow j+1$; $j \leftarrow j+1$; $i \leftarrow i+1$;

w przeciwnym razie, jeżeli $j = 0$, //jeżeli $BS[i]$ nie ma nawet jednoliterowego presufiksu

to $F[i+1] \leftarrow 0$; $i \leftarrow i+1$

w przeciwnym razie $j \leftarrow F[j]$ //zgodnie z lem. 6.1, $S[0]S[1]\ldots S[F[j]-1]$ jest kolejnym, krótszym presufiksem B

//cała tablica F została wypełniona i jest do wykorzystania.

6.4 Złożoność algorytmu

Można zauważyć, że każde wykonanie pętli **dopóki** zwiększa wartość wyrażenia $2i - j$. Początkowa wartością tego wyrażenia jest 2, końcowa – nie przekracza $2|\ S\ |$. Pętla jest więc wykonywana najwyżej $2|\ S\ |$ razy i złożoność algorytmu jest $O(|\ S\ |)$.

Aby zrozumieć ten dowód, można się przyjrzeć raz jeszcze algorytmowi. Polega on sprawdzaniu coraz to innych wartości i, j, czy prefiks $S[0]S[1]\ldots S[j]$ jest sufiksem $S[0]S[1]\ldots S[i]$. Jest to także sprawdzanie, czy sufiks $S[i-j]S[i-j+1]\ldots S[i]$ słowa $S[0]S[1]\ldots S[i]$ jest także jego prefiksem. Podczas wykonania algorytmu ten sufiks przesuwa się stopniowo w prawo. I to podczas każdego wykonania pętli przesuwa się istotnie: zwiększa się albo indeks lewego końca, albo obu końców o tę samą wartość. Zwiększa się więc suma indeksów obu końców wspomnianego sufiksu.

7 Kolejny algorytm rozwiązujący zadanie 8

Algorytm ten będzie bardzo podobny do poprzedniego. Wcześniej, dla każdego prefiksu słowa S szukaliśmy możliwie długiego, właściwego sufiksu, który był także prefiksem S. Teraz dla każdego prefiksu słowa T będziemy szukać jego możliwie długiego sufiksu, który jest także prefiksem słowa S i zliczać te prefiksy T, które będą się kończyć całym słowem S.

Przypomnijmy specyfikację zadania:

Dane: dwa słowa: niepuste słowo S i słowo (tekst) T.

Wynik: liczbna wystąpień słowa S w słowie T.

Specyfikacja ta wymaga uzupełnienia. Słowo S powinno być znane w całości, w szczególności powinna być znana jego długość i nie powinna być to zbyt duża liczba. Słowo T może być czytane litera po literze, jego długość nie musi być znana i może być bardzo duża. Może być to słowo, którego nie można zapamiętać w pamięci operacyjnej komputera. Musimy jednak umieć stwierdzić, że zostało przeczytane całe słowo T. Jedno z rozwiązań, które jest w tym celu stosowane, przewiduje, że każde słowo jest zakończone specjalnym znakiem końca słowa, czymś w rodzaju kropki kończącej zdanie. Będziemy zakładać, że słowo T kończy się takim specjalnym znakiem.

Interesujące nas zadanie może zostać rozwiązane za pomocą następującego algorytmu:

Dane: dwa słowa: niepuste słowo S i tekst T.

Uruchom algorytm z rozdziału 6 podając jako dane słowo S i oblicz tablicę F;

//dla przypomnienia: tablica F jest indeksowana od 1 do $|S|$, element $F[i]$ zawiera długość maksymalnego presufiksu słowa $S[0]S[1]\ldots S[i-1]$ (prefiksu długości i słowa S).

$l \leftarrow 0$; //zmienna l będzie pamiętać liczbę dotychczas znalezionych wystąpień słowa S.

$j \leftarrow 0$; //j to indeks wskazujący literę (prefiksu) słowa S, początkowo wskazuje pierwszą literę S.

wczytaj t; //dokładniej: instrukcja ta przypisuje zmiennej t pierwszą, nieczytaną jeszcze literę słowa T. Jeżeli tekst T jest pusty (lub został cały przeczytany), to instrukcja ta spowoduje przypisanie zmiennej t specjalnego znaku końca tekstu.

$i \leftarrow 1$; //Zmienna ta pamięta liczbę pobranych liter słowa T. Nie jest potrzebna, można ją usunąć z algorytmu, ale zostanie wykorzystana do oceny złożoności.

dopóki t nie jest znakiem końca tekstu **wykonaj**

//w tym momencie słowo $A = S[0]S[1]\ldots S[j-1]$ jest sufiksem słowa $B = T[0]T[1]\ldots T[i-2]$, a t jest kolejnym (i-tym) znakiem słowa T, $t = T[i-1]$.

jeżeli $t = S[j]$, //jeżeli $AS[j]$ jest sufiksem Bt

to wczytaj t; $i \leftarrow i+1$; $j \leftarrow j+1$; //przygotowujemy się do badania kolejnych znaków słów S i T, wartością j jest teraz $|\,A\,|+1$, czyli długość znalezionego sufiksu.

jeżeli $j = |\,S\,|$, //jeżeli znaleziony sufiks jest całym S

to $l \leftarrow l+1$; $j \leftarrow F[j]$ //zwiększamy licznik wystąpień S i przygotowujemy się do badania, czy najdłuższy presufiks znalezionego wystąpienia S daje się wydłużyć do kolejnego wystąpienia

w przeciwnym razie, jeżeli $j = 0$, //jeżeli wszystkie prefiksy S kończące słowo B, nawet pusty, nie są częścią wystąpienia S

to wczytaj t; $i \leftarrow i+1$ //wydłużamy badany fragment słowa T, wiemy, że nie kończy się on żadnym niepustym prefiksem S, zauważmy, że wartość j nadal jest równa 0

w przeciwnym razie $j \leftarrow F[j]$ //$S[0]S[1]\ldots S[F[j]-1]$ jest kolejnym, krótszym sufiksem B, który być może jest częścią kolejnego wystąpienia S, przystępujemy do sprawdzania, czy tak jest.

wypisz l.

7.1 Ocena złożoności przedstawionego algorytmu

Złożoność powyższego algorytmu szacujemy dokładnie tak, jak algorytmu z rozdziału 6: każdorazowe wykonanie instrukcji z pętli **dopóki** powoduje zwiększenie wartości wyrażenia $2i-j$. Początkowo wartość ta jest równa 2. Zmienna i zlicza wczytane litery słowa T, tak więc instrukcje w tej pętli są wykonywane najwyżej $2|\,T\,|$ razy. Pierwsza instrukcja algorytmu wymaga wykonania $O(|\,S\,|)$ czynności, pozostałe – $O(|\,T\,|)$. Złożoność całego algorytmu jest więc $O(|\,S\,|+|\,T\,|)$.