Wojciech Rytter	Jakub Radoszewski	Marek Cygan
Treść zadania, Opracowanie	Opracowanie, Program	Program

OI, Etap II, dzień 2, 10–02–2005

Szablon

Bajtazar chce umieścić na swoim domu długi napis. W tym celu najpierw musi wykonać odpowiedni szablon z wyciętymi literkami. Następnie przykłada taki szablon we właściwe miejsce do ściany i maluje po nim farbą, w wyniku czego na ścianie pojawiają się literki znajdujące się na szablonie. Gdy szablon jest przyłożony do ściany, to malujemy od razu wszystkie znajdujące się na nim literki (nie można tylko części). Dopuszczamy natomiast możliwość, że któraś litera na ścianie zostanie narysowana wielokrotnie, w wyniku różnych przyłożeń szablonu. Literki na szablonie znajdują się koło siebie (nie ma tam przerw). Oczywiście można wykonać szablon zawierający cały napis. Bajtazar chce jednak zminimalizować koszty i w związku z tym wykonać szablon tak krótki, jak to tylko możliwe.

Zadanie

Napisz program który:

- *wczyta ze standardowego wejścia napis, który Bajtazar chce umieścić na domu,*
- *obliczy minimalną długość potrzebnego do tego szablonu,*
- *wypisze wynik na standardowe wyjście.*

Wejście

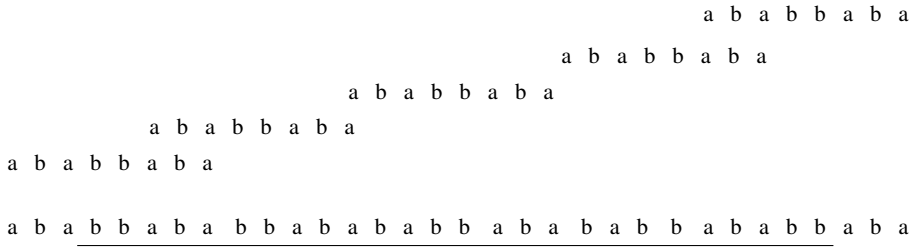
W pierwszym i jedynym wierszu standardowego wejścia znajduje się jedno słowo. Jest to napis, który Bajtazar chce umieścić na domu. Napis składa się z nie więcej niż 500 000, oraz nie mniej niż 1 małej litery alfabetu angielskiego.

Wyjście

W pierwszym i jedynym wierszu standardowego wyjścia należy zapisać jedną liczbę całkowitą — minimalną liczbę liter na szablonie.

Przykład

Dla danych wejściowych:
```
ababbababbababbababababbababbaba
```
poprawnym wynikiem jest:
```
8
```



Rysunek pokazuje, że szablon ababbaba może służyć do namalowania napisu z przykładu.

Rozwiązanie

Problem występujący w zadaniu jest związany z zagadnieniem wyszukiwania wzorca w tekście. Napis, który Bajtazar chce umieścić na domu, pełni rolę *tekstu* a szablon wykorzystywany do malowania to *wzorzec*. W zadaniu zależy nam na znalezieniu wzorca występującego odpowiednio gęsto w tekście (by szablon pokrył cały napis). Dodatkowo procedura wyszukiwania wzorca musi być odpowiednio efektywna, z racji dopuszczalnego rozmiaru danych.

Najprostsze rozwiązanie — złożoność czasowa $O(n^3)$

Niech x będzie napisem Bajtazara i niech $n = |x|$ będzie długością słowa x. Istnieje wiele algorytmów rozwiązania zadania o różnych złożonościach czasowych. Na początek omówimy najprostszy (a zarazem najwolniejszy) z nich.

Algorytm ten polega na sprawdzeniu wszystkich podsłów słowa x i wybraniu najkrótszego z nich, będącego szablonem dla x. Wszystkich podsłów słowa x jest $O(n^2)$. „Naiwne" sprawdzanie dla każdej pozycji w słowie x, czy występuje na niej słowo y, wymaga czasu $O(|y| \cdot n)$. Potem wystarczy sprawdzić, czy pomiędzy znalezionymi wystąpieniami nie ma luki większej niż długość słowa y. W ten sposób dostajemy algorytm działający w czasie $O(n^4)$.

W miejsce algorytmu naiwnego możemy zastosować szybszy algorytm wyszukiwania wzorca w tekście. Na przykład działający w czasie liniowo zależnym od sumy długości wzorca i tekstu (w tym przypadku w czasie $O(n)$) algorytm Knutha-Morrisa-Pratta, patrz rozdział 5 w [14]. Stosując taki algorytm uzyskujemy złożoność $O(n^3)$.

Szybszy algorytm — złożoność czasowa $O(n^2)$

Definicja 1 Dla słowa x powiemy, że y jest jego *prefiksem*, jeśli $y = x[1..i]$ dla pewnego $1 \leqslant i \leqslant n$. Powiemy, że y jest *sufiksem* słowa x, jeśli $y = x[i..n]$ dla pewnego $1 \leqslant i \leqslant n$. Prefiks (analogicznie sufiks) nazwiemy *właściwym*, jeśli jego długość jest mniejsza niż długość x.

Na potrzeby zadania wprowadzimy także pojęcie *prefikso-sufiksu* x — słowa, które jest jednocześnie prefiksem i sufiksem x.

Podstawowa prosta obserwacja, na której oprzemy rozwiązanie zadania, jest następująca.

Obserwacja 1 *Szablon napisu musi być jego prefikso-sufiksem.*

Oznaczmy przez $\mathcal{S}$ zbiór prefikso-sufiksów napisu x, które mogą potencjalnie być szablonami dla x. Aby go dokładnie opisać wykorzystamy pojęcie tablicy sufiksowej.

Definicja 2 Tablicę $P[1..n]$ nazwiemy *tablicą sufiksową* słowa x, jeśli

$$P[i] = \max\{\, j \mid 0 \leqslant j < i \text{ oraz } x[1..j] = x[i-j+1..i]\} \quad \text{dla } 1 \leqslant i \leqslant n,$$

co oznacza, że $x[1..P[i]]$ jest najdłuższym właściwym prefikso-sufiksem słowa $x[1..i]$.

Istnieją algorytmy wyliczania tablicy sufiksowej w czasie liniowo zależnym od długości słowa x. Czytelnik może się z nimi zapoznać na przykład w [14]. My natomiast możemy już sformułować lemat dokładnie charakteryzujący zbiór $\mathcal{S}$.

Lemat 2

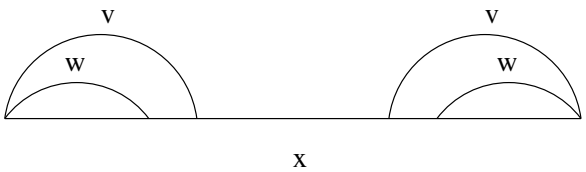
$$\mathcal{S} \;=\; \Big\{x[1..n],\; x[1..P[n]],\; x[1..P[P[n]]],\; \ldots\, x[1..P^k[n]]\Big\}$$

gdzie $P^k[n]$ *oznacza* $\underbrace{P[P[\cdots P[n]\cdots]]}_{k \text{ razy}}$, *a* k *jest dobrane tak, by* $P^k[n] > 0$ *oraz* $P^{k+1}[n] = 0$.

Dowód Zauważmy, że:

(i) jeżeli słowa v i w są prefikso-sufiksami słowa x oraz $|w| \leqslant |v|$, to słowo w jest prefikso-sufiksem słowa v (patrz rysunek);

(ii) jeżeli słowo w jest prefikso-sufiksem słowa v i słowo v jest prefikso-sufiksem słowa x, to w jest prefikso-sufiksem x (także patrz rysunek).



Prawdziwość lematu wykażemy przez indukcję względem długości słowa x. Dla $|x| = 1$ lub $P[n] = 0$ zbiór $\mathcal{S}$ zawiera tylko słowo x i lemat jest oczywiście prawdziwy.

Dla $|x| > 1$ i $P[n] > 0$ niech $y = x[1..P[n]]$ będzie najdłuższym właściwym prefikso-sufiksem x. Zauważmy, że dla zbioru $\mathcal{S}'$ utworzonego dla y analogicznie jak $\mathcal{S}$ dla x, zachodzi równość

$$\mathcal{S} = \mathcal{S}' \cup \{x\}.$$

Ponadto dla słowa y zachodzi założenie indukcyjne mówiące, że zbiór $\mathcal{S}'$ to wszystkie prefikso-sufiksy tego słowa.

Aby wykazać, że wszystkie elementy zbioru $\mathcal{S}$ są prefikso-sufiksami x zauważmy, że x jest w sposób oczywisty swoim prefikso-sufiksem. Także y jest prefikso-sufiksem x i z założenia indukcyjnego wiemy, że wszystkie elementy $\mathcal{S}'$ są prefikso-sufiksami słowa y, a więc z własności (ii) są także prefikso-sufiksami słowa x.

By pokazać, że każdy prefikso-sufiks x należy do zbioru $\mathcal{S}$, najpierw zauważmy, że $x \in \mathcal{S}$. Pozostałe prefikso-sufiksy słowa x są nie dłuższe niż y, więc z własności (i) widzimy, że są prefikso-sufiksami y i z założenia indukcyjnego należą do zbioru $\mathcal{S}' \subset \mathcal{S}$.

■

Teraz możemy zapisać algorytm poszukiwania szablonu, który sprawdzi każde słowo $y \in \mathcal{S}$, począwszy od najkrótszych. Sprawdzenia dokonujemy, jak poprzednio, za pomocą liniowego algorytmu wyszukiwania wzorca w tekście. Otrzymujemy algorytm działający w czasie $O(n^2)$, gdyż moc zbioru $\mathcal{S}$ jest ograniczona przez n (chociaż zbiór ten jest zazwyczaj znacznie mniejszy).

Algorytm o złożoności czasowej $O(n \log n)$

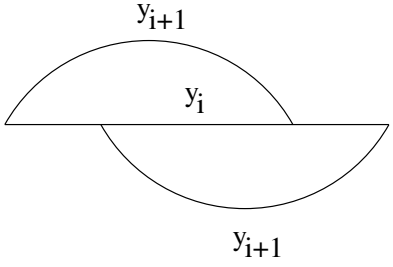
Okazuje się, że liczbę testowanych szablonów możemy istotnie ograniczyć dzięki następującemu spostrzeżeniu:

Obserwacja 3 *Niech $y_1, y_2, \ldots, y_k$ będą elementami zbioru $\mathcal{S}$ wypisanymi w takiej kolejności, że $|y_1| > |y_2| > \cdots > |y_k|$. Wówczas, jeśli dla pewnego $1 \leqslant i < k$ zachodzi nierówność*

$$\frac{|y_i|}{2} \leqslant |y_{i+1}|, \tag{1}$$

to słowo y_i nie jest najkrótszym szablonem dla napisu x.

Dowód Niech y_i oraz y_{i+1} będą elementami $\mathcal{S}$ spełniającymi powyższą nierówność. Jeśli y_i nie jest szablonem dla x, to oczywiście teza jest prawdziwa. Jeśli natomiast y_i jest szablonem dla x, to krótsze słowo y_{i+1} również nim jest, gdyż w całości pokrywa y_i, a tym samym x.



■

W ten sposób poszukując najkrótszego szablonu dla napisu x możemy rozważać tylko niektóre elementy zbioru $\mathcal{S}$. Zaczynamy od najkrótszego $y = y_k$ i jeśli nie jest on szablonem, to jako kolejnego kandydata wybieramy najkrótsze słowo z $\mathcal{S}$ o długości większej niż $2|y|$. W najgorszym razie musimy więc sprawdzić $\log n$ słów y.

Czas działania takiego algorytmu jest rzędu $O(n \log n)$ i program napisany na podstawie tego algorytmu przechodzi wszystkie testy dla zadania.

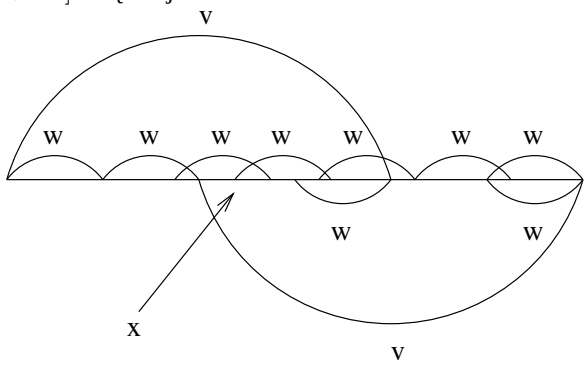
Algorytm o złożoności czasowej $O(n)$

Okazuje się, że istnieje jeszcze szybszy algorytm — pozwalający rozwiązać zadanie w czasie *liniowo* zależnym od rozmiaru danych. Polega on na wyliczaniu kolejno dla $i = 1, 2, \ldots, n$ wartości *Szablon*$[i]$ — długości najkrótszego szablonu dla słowa $x[1 \ldots i]$.

W algorytmie istotnie wykorzystujemy własność pokazaną w następującym lemacie:

Lemat 4 *Jeżeli słowo v jest prefikso-sufiksem słowa x i słowo w jest szablonem dla słowa x oraz $|w| \leqslant |v|$, to słowo w jest także szablonem dla słowa v.*

Dowód Niech $v = x[1..i]$. Weźmy tylko te wystąpienia szablonu w w napisie x, które w całości mieszczą się w słowie v. Zauważmy, że muszą one pokrywać słowo $v[1..i-|w|]$, bo wystąpienia w wystające poza v muszą zaczynać się na pozycjach większych niż $i-|w|+1$. Ponadto, ponieważ w jest także prefikso-sufiksem v (z własności (i) w dowodzie lematu 2), więc $w = v[i-|w|+1..i]$ i stąd w jest szablonem dla v.



■

W kolejnym lemacie prezentujemy główną ideę pozwalającą rozwiązać problem w czasie liniowym.

Lemat 5 *Jeżeli najkrótszy szablon słowa $v = x[1..P[n]]$ nie jest szablonem słowa $x = x[1..n]$, to słowo x ma tylko jeden szablon i jest nim całe słowo x.*

Dowód Przeprowadzimy dowód nie wprost. Oznaczmy przez s najkrótszy szablon słowa v.

Załóżmy nie wprost, że s nie jest szablonem słowa x i istnieje szablon s' słowa x, dla którego $s' \neq x$.

Rozważmy następujące przypadki, w zależności od długości słowa s':

1. Jeżeli $|s'| > |v|$, to mamy natychmiastową sprzeczność, gdyż s' — jako szablon x — jest jego prefikso-sufiksem i nie może być dłuższy od v — najdłuższego prefikso-sufiksu słowa x.

2. Jeżeli $|s| < |s'| \leqslant |v|$, to z lematu 4 (zastosowanego dla słów s', v i x) s' jest szablonem v, a więc jego prefikso-sufiksem. Ponownie wykorzystując lemat 4, tym razem dla słów s, s' i v, mamy, że słowo s jest szablonem słowa s'. Stąd jeżeli słowo x może być pokryte słowem s' jako szablonem, to może być także pokryte słowem s jako szablonem, co jest sprzeczne z założeniami lematu.

3. Jeżeli $|s'| < |s|$, to z lematu 4 mamy, że s' jest szablonem v, a to stanowi sprzeczność z założeniem, że s jest najkrótszym szablonem v.

■

Algorytm

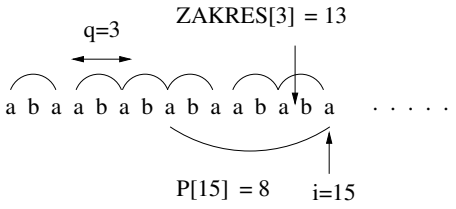
Dla każdej pozycji i w słowie x obliczymy najkrótszy szablon słowa $x[1..i]$. W tym celu sprawdzimy najkrótszy szablon słowa $x[1 \ldots P[i]]$. Jeśli jest to szablon słowa $x[1..i]$, to jest to także najkrótszy szablon $x[1..i]$. W przeciwnym razie najkrótszym szablonem $x[1..i]$ jest $x[1..i]$.

Aby przyśpieszyć sprawdzanie powyższego warunku, w dodatkowej tablicy $Zakres[1 \ldots n]$ na pozycji j będziemy przechowywać informacje o tym, jaki prefiks słowa x można pokryć słowem $x[1 \ldots j]$ jako szablonem, czyli $Zakres[j] = k$ oznacza, iż słowo $x[1 \ldots j]$ jest szablonem dla słowa $x[1 \ldots k]$.

Przyjmijmy początkowo $Zakres[i] = i$, $Szablon[i] = i$, dla $0 \leqslant i \leqslant n$.

```
 1: procedure  Algorytm liniowy
 2:  begin
 3:    for i:=2 to n do
 4:      s:=Szablon[P[i]];
 5:      if P[i]> 0 and Zakres[s]⩾i−s then { warunek (1) }
 6:        begin
 7:          Szablon[i]:=s;
 8:          Zakres[s]:=i;
 9:        end
10:     return Szablon[n];
11:  end
```

Przykład



Iteracja algorytmu dla $i = 15$ oraz słowa $x = abaababababaababa\ldots$. Tuż przed rozpoczęciem tej iteracji mamy $P[i] = 8$, $q = Szablon[8] = 3$, $Zakres[3] = 13$. Rozważamy więc maksymalny prefikso-sufiks x, czyli $x[1..8]$ — da się go pokryć szablonem $x[1..Szablon[8]]$. Tym szablonem (długości 3) da się także pokryć lukę $x[Zakres[3]+1..i]$. Stąd po zakończeniu iteracji mamy $Szablon[15] = 3$, $Zakres[3] = 15$.

Dowód poprawności rozwiązania

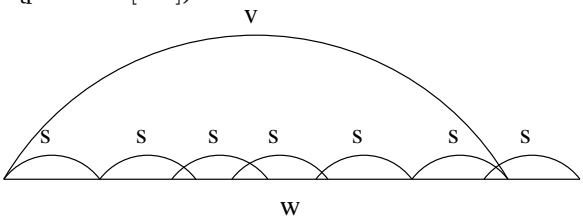
Pokażemy przez indukcję względem i, że warunek (1) instrukcji **if** w i-tej iteracji pętli **for** jest równoważny następującemu warunkowi:

najkrótszy szablon słowa x[1...P[i]] jest szablonem słowa x[1...i]. (2)

Dla $i = 1$ żaden z warunków nie zachodzi. Załóżmy zatem, że $i > 1$.

Po pierwsze zauważmy, że z założenia indukcyjnego wynika, że dla wszystkich wartości $q < i$, jeśli $Zakres[q] = k$, to $x[1..q]$ jest rzeczywiście szablonem $x[1..k]$, bo tylko wtedy, gdy zachodzi warunek (1) jest nadawana nowa wartość tabeli *Zakres*. Stąd warunek (1) implikuje warunek (2).

Przyjmijmy, że $Szablon[P[i]] = s$. Wystarczy pokazać, że jeżeli słowo $x[1..s]$ jest szablonem $x[1..i]$, to zachodzi warunek (1), czyli $Zakres[s] \geqslant i - s$. W tym celu zauważmy, że istnieje prefiks właściwy $x[1..j]$ słowa $x[1..i]$, o długości większej lub równej $i - s$ (właściwy, czyli $j < i$), którego szablonem jest $x[1..s]$ (wystarczy wziąć $x[1..i]$ bez końcówki pokrytej tylko ostatnim wystąpieniem $x[1..s]$):



Gdyby dla $x[1..j]$ istniał szablon $x[1..s']$ krótszy niż $x[1..s]$, to stosując lemat 4 dla $x[1..s']$, $x[1..s]$ (który jako szablon jest prefikso-sufiksem $x[1..j]$) oraz $x[1..j]$ mielibyśmy, że $x[1..s']$ jest szablonem $x[1..s]$, a więc i szablonem słowa $x[1...P[i]]$. To z kolei stałoby w sprzeczności z tym, że $x[1..s]$ jest najkrótszym szablonem $x[1...P[i]]$.

Stąd $x[1..s]$ jest najkrótszym szablonem słowa $x[1..j]$, a więc w j-tej iteracji pętli ustaliliśmy $Zakres[s] = j$, a potem mogliśmy tę wartość już tylko zwiększyć. Czyli przy założeniu, że zachodzi warunek (2), warunek (1) także jest spełniony, co należało pokazać.

Rozwiązania wzorcowe

Na dysku dołączonym do książeczki w plikach `sza.pas` i `sza1.cpp` jest zaimplementowany algorytm działający w czasie $O(n \log n)$. Implementacja rozwiązania działającego w czasie $O(n)$ znajduje się w plikach `sza2.pas` i `sza3.cpp`.

Testy

Rozwiązania zawodników były sprawdzane na 10 testach.

Nazwa	n	k	l	Opis
sza1.in	42	4	5	aab/aaab
sza2.in	61	4	9	aabcc
sza3.in	11 168	623	3 723	aab
sza4.in	12 611	318	3 153	aaabb
sza5.in	22 816	385	3 805	yyyyx
sza6.in	437 214	2 496	19 875	aabc
sza7.in	420 008	35 002	140 003	ab
sza8.in	495 019	5 505	99 005	aaaabcdcb
sza9.in	495 020	906	99 006	aaaaabcd...zz...b
sza10.in	494 211	123 555	247 105	a

W powyższym opisie n oznacza długość napisu, k to liczba prefikso-sufiksów, a l to długość szablonu. Słowa podane w opisach to fragmenty napisów, które najczęściej w nich występują. Testy były konstruowane poprzez wielokrotne powtarzanie tych fragmentów, sporadycznie rozdzielanych innymi, krótkimi słowami.

Aby rozróżnić poszczególne rozwiązania testy musiały zawierać dużą liczbę prefikso-sufiksów, co spowodowało, że mają one w większości bardzo regularną strukturę (losowy test z bardzo dużym prawdopodobieństwem ma tylko jeden szablon, będący całym napisem). Ponadto w większości testów użyta jest mała liczba różnych liter, gdyż zwiększa to liczbę prefikso-sufiksów.