

Zadanie: potęgi słów

Dla słowa s , przez $|s|$ oznaczamy długość s .

Okres słowa s to najmniejsze dodatnie p takie, że $s[i]=s[i+p]$ dla każdego $i = 0, \dots, |s| - p - 1$. Przykładowo: okres słowa $abcb$ to 3, okres słowa $ababab$ to 2, okres słowa xyz to 3.

Przez k -tą potęgę słowa x rozumiemy k -krotne powtórzenie słowa x . Przykładowo: $abcbcbcb$ to trzecią potęgą słowa abc .

Zadanie składa się z dwóch części.

1. Wyznaczenie okresu zadanego słowa (metoda Period). Uwaga: metoda musi działać w czasie $O(|s|)$
2. Wyznaczenie największej potęgi zawartej w danym słowie (metoda MaxPower).

Metoda powinna dla zadanego słowa s zwrócić największe k takie, że k -ta potęga jakiegoś słowa jest zawarta w s jako podślowo. Przykładowo, dla słowa $s=abxyzxyzcd$ odpowiedzią powinno być 2, bo s zawiera drugą potęgę xyz jako podślowo.

Ponadto, w parametrach wyjściowych należy przekazać miejsce wystąpienia znalezionej potęgi, tzn. `startIndex` to indeks pierwszej litery znalezionej potęgi, a `endIndex` to indeks pierwszej litery znajdującej się za znaną potęgą. W powyższym przykładzie, `startIndex=2`, a `endIndex=8`

Uwaga: metoda musi działać w czasie $O(|s|^2)$

Wskazówki:

- Wykorzystaj algorytm KMP.
- W części 1. wystarczy znaleźć najdłuższe słowo będące jednocześnie właściwym prefiksem i sufiksem s .
- W części 2. należy wyznaczyć najdłuższe słowo będące jednocześnie właściwym prefiksem i sufiksem każdego z podśłów słowa s , a następnie sprawdzić, które z tych podśłów są potęgami (a słowo jest potęgą, jeśli jego długość jest podzielna przez okres). Naiwna implementacja będzie miała złożoność $O(|s|^3)$, ale łatwo to poprawić: wystarczy zauważyć, że raz policzona tablica P (z algorytmu KMP) może być użyta dla wielu podśłów słowa s .

Punktacja:

- Etap 1: 0.5pkt
- Etap 2: 1.5 pkt za prawidłową odpowiedź + 0.5 pkt za prawidłowe `startIndex` i `endIndex`

Zadanie: kompresja bezstratna algorytmem LZ77

Algorytm LZ77¹ koduje tekst w postaci ciągu trójek (p_i, c_i, s_i) , gdzie p_i i c_i są nieujemnymi liczbami całkowitymi, a s_i pojedynczym znakiem. Trójki te stanowią instrukcję, jak można odtworzyć oryginalny tekst: w i -tym kroku należy przepisać z dotychczas odkodowanego fragmentu tekstu c_i kolejnych znaków, poczynając od znaku znajdującego się p_i pozycji przed ostatnim odkodowanym znakiem, a następnie dopisać jeszcze znak s_i . Zauważmy, że $p_i + 1$ może być mniejsze niż c_i , wtedy należy przepisać również znaki dopisane w tym samym kroku.

Przykład: Ciąg $(0,0,a), (0,1,b), (1,3,c)$ koduje tekst aababac (po pierwszym kroku mamy słowo a; w drugim kroku przepisujemy ostatni znak i dopisujemy b otrzymując aab, w trzecim kroku przepisujemy trzy znaki poczynając od drugiego wystąpienia litery a i dopisujemy c).

Pierwsza część zadania polega na zdekodowaniu tekstu zadokowanego w taki sposób, jak opisano powyżej.

Celem drugiej części zadania, dla zadanego tekstu t oraz stałej p_{max} , jest znalezienie kodowania t , w którym dla każdego i zachodzi $p_i \leq p_{max}$ oraz liczba użytych trójek jest minimalna. Można to robić odwracając powyższą procedurę dekodowania i działając zachłannie, tzn. w każdym kroku dodawać do wyniku trójkę (p_i, c_i, s_i) z największym możliwym c_i .

Dokładniej, algorytm kodujący w każdym kroku powinien wykonywać następujące operacje:

1. Niech w = ostatnie $p_{max} + 1$ zakodowanych znaków z t
2. Niech r = wszystkie niezakodowane znaki z t
3. Znajdź w słowie wr najdłuższe możliwe wystąpienie właściwego prefiksu słowa r zaczynające się wśród pierwszych $|w|$ pozycji. Niech c będzie długością znalezionej prefiksu a j indeksem pierwszego znaku znalezionej prefiksu liczonym od 1.
4. Dodaj do wyniku trójkę $(|w| - j, c, r[c + 1])$ ²
5. Oznacz pierwsze $c + 1$ znaków z r jako zakodowane

Na maksymalną ocenę z części laboratoryjnej wymagana jest implementacja, w której każdy krok będzie realizowany w czasie liniowym (tzn. $O(n)$, gdzie n jest liczbą znaków w kodowanym napisie). Można otrzymać zmniejszoną liczbą punktów za implementację naiwną.

W części domowej wymagana jest efektywniejsza implementacja, która w kroku i będzie wykonywała $O(p_{max} + c_i)$ operacji (zauważmy, że może to być dużo mniej niż $O(n)$).

Punktacja:

- Dekodowanie – 0.5p
- Kodowanie naiwne – 0.5p
- Kodowanie, w którym każdy krok wykonuje się w czasie $O(t) - 2p$

Wskazówki:

- Krok 3. można zrealizować modyfikując algorytm KMP. Wystarczy traktować r jako wzorzec i zapamiętywać, kiedy udało się dopasować najdłuższy fragment wzorca.
- Najlepszą złożoność $O(p_{max} + c)$ można osiągnąć wyliczając tablicę P w algorytmie KMP w sposób leniwy (wyznaczać kolejną wartość dopiero, kiedy jest potrzebna) i odwołując się do słów w i r przez odpowiednie indeksowanie oryginalnego tekstu, bez zbędnego kopiowania.

¹W zadaniu opisana jest pewna modyfikacja algorytmu – pomijamy górne ograniczenie na długość dopasowywanego wzorca. W praktycznych zastosowaniach (na przykład w formacie .zip) to ograniczenie jest istotne i, zwykle, dużo mniejsze niż długość fragmentu, w którym tego wzorca szukamy.

²Znaki w r indeksujemy od 1; np. dla $c=2$, $r[c+1]$ jest trzecim znakiem z r .