**Jakub Radoszewski, Wojciech Rytter** — Treść zadania, Opracowanie
**Piotr Niedźwiedź** — Program

**Dostępna pamięć: 64 MB.** *OI, Etap II, dzień pierwszy, 10.02.2010*

# Chomiki

*Bajtazar prowadzi hodowlę chomików. Każdy chomik ma unikalne imię, złożone z małych liter alfabetu angielskiego. Chomiki mają obszerną i komfortową klatkę. Bajtazar chce umieścić pod klatką wyświetlacz, który będzie wyświetlał imiona jego chomików. Wyświetlacz będzie miał postać ciągu liter, z których każda może być zapalona lub zgaszona. Naraz będzie wyświetlane tylko jedno imię chomika. Zapalone litery tworzące to imię muszą znajdować się obok siebie.*

*Bajtazar chce, aby wyświetlacz mógł wyświetlać imiona chomików przynajmniej w $m$ różnych miejscach. Dopuszcza on, aby to samo imię mogło być wyświetlane w kilku różnych miejscach, i nie wymaga, aby każde imię mogło być wyświetlane na wyświetlaczu. Zauważ, że wystąpienia imion na wyświetlaczu mogą dowolnie na siebie nachodzić. Możesz założyć, że imię żadnego z chomików nie występuje (jako spójny fragment) w imieniu żadnego innego chomika. Bajtazar poprosił Cię o pomoc w wyznaczeniu najmniejszej liczby liter, z jakich musi składać się wyświetlacz.*

*Inaczej mówiąc, należy wyznaczyć minimalną długość napisu (złożonego z małych liter alfabetu angielskiego), w którym łączna liczba wystąpień imion chomików jest nie mniejsza niż $m$. (Mówimy, że słowo $s$ występuje w napisie $t$, jeżeli $s$ stanowi spójny fragment $t$).*

## Wejście

*Pierwszy wiersz standardowego wejścia zawiera dwie liczby całkowite $n$ oraz $m$ ($1 \leqslant n \leqslant 200$, $1 \leqslant m \leqslant 10^9$), oddzielone pojedynczym odstępem i oznaczające, odpowiednio, liczbę chomików Bajtazara i minimalną liczbę wystąpień imion chomików na wyświetlaczu. Każdy z kolejnych $n$ wierszy zawiera niepusty napis złożony z małych liter alfabetu angielskiego będący imieniem chomika. Sumaryczna długość wszystkich imion nie przekracza $100\ 000$ liter.*

## Wyjście

*Pierwszy i jedyny wiersz standardowego wyjścia powinien zawierać jedną liczbę całkowitą — minimalną liczbę liter, z których musi być zbudowany wyświetlacz.*

## Przykład

*Dla danych wejściowych:*
```
4 5
monika
tomek
szymon
bernard
```

*poprawnym wynikiem jest:*
```
23
```

*Najkrótszy wyświetlacz może mieć, na przykład, postać:* `szymonikatomekszymonika`*. Zawiera on łącznie 5 wystąpień imion chomików:* `szymon` *i* `monika` *występują dwukrotnie,* `tomek` *raz, a* `bernard` *ani razu.*

# Rozwiązanie

## Trochę formalizmów

Na początku opisu wprowadźmy dla wygody kilka oznaczeń. W tym zadaniu interesować nas będą słowa złożone z małych liter alfabetu angielskiego. Jeśli $u$ jest takim słowem, to przez $|u|$ oznaczymy jego długość, czyli liczbę liter, z których się składa. Powiemy, że słowo $u$ jest *prefiksem* (*sufiksem*, *podsłowem*) słowa $v$, jeżeli słowo $u$ stanowi początkowy fragment (odpowiednio końcowy fragment albo dowolny spójny fragment) słowa $v$. Przez $\#wyst(u,v)$ oznaczymy łączną liczbę wystąpień słowa $u$ w postaci podsłów słowa $v$. Przykładowo, słowo `aba` jest zarazem prefiksem i sufiksem słowa `ababaaaba` oraz $\#wyst($`aba`, `ababaaaba`$) = 3$.

Niech $S = \{s_1, s_2, \ldots, s_n\}$ będzie zbiorem $n$ słów reprezentujących imiona chomików Bajtazara. W tym zadaniu poszukujemy najkrótszego słowa $t$, w którym łączna liczba wystąpień imion chomików jest nie mniejsza niż $m$, co przy wprowadzonych oznaczeniach możemy zapisać jako:

$$\#wyst(s_1,t) + \#wyst(s_2,t) + \ldots + \#wyst(s_n,t) \;\geqslant\; m. \qquad (1)$$

Warto dodać, że parametr $m$ może być rzędu $10^9$, czyli możemy być zmuszeni do poszukiwania naprawdę długiego słowa $t$. To oznacza, że w rozwiązaniu powinniśmy ograniczyć się do wyznaczenia długości słowa $t$, bez konstruowania go.

Zakładamy dalej, że słowa ze zbioru $S$ są niepuste ($|s_i| > 0$) i mają łącznie długość $L$, tzn. $L = |s_1| + |s_2| + \ldots + |s_n|$. W zadaniu zaszyte jest jeszcze jedno, dosyć tajemnicze ograniczenie, że żadne ze słów $s_i$ nie stanowi podsłowa żadnego innego, tj.:

$$\#wyst(s_i, s_j) = 0 \quad \text{dla } i \neq j. \qquad (2)$$

## Jak wygląda rozwiązanie?

Spróbujmy przyjrzeć się temu, jaką strukturę powinno mieć wynikowe słowo $t$. Niech $s_{i_1}, s_{i_2}, \ldots, s_{i_m}$ będzie sekwencją reprezentującą pierwsze (od lewej) $m$ wystąpień słów ze zbioru $S$ w słowie $t$. Dodajmy dla jasności, że na każdej pozycji słowa $t$ może rozpoczynać się co najwyżej jedno wystąpienie jakiegoś ze słów $s_{i_j}$, gdyż w przeciwnym przypadku któreś ze słów ze zbioru $S$ byłoby prefiksem któregoś innego, co nie jest możliwe wobec warunku (2).

Zacznijmy od tego, że słowo $s_{i_1}$ musi być prefiksem słowa $t$. Faktycznie, gdyby tak nie było, to słowo powstałe z $t$ przez wyrzucenie pierwszej litery nadal spełniałoby warunek (1), a więc $t$ nie byłoby najkrótsze.

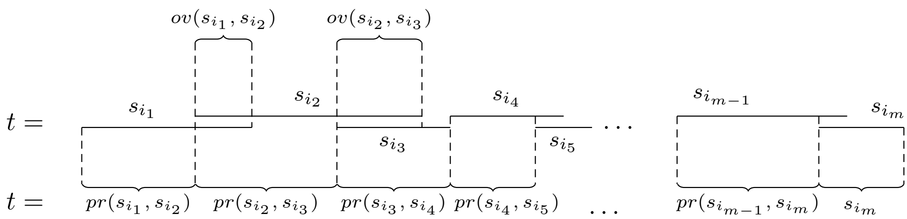
Następne w słowie $t$ występuje $s_{i_2}$. Łatwo widać, że jego wystąpienie musi zaczynać się w ramach pierwszych $|s_{i_1}| + 1$ liter słowa $t$, gdyż w przeciwnym przypadku znów moglibyśmy wyciąć jedną literę słowa $t$, nie zaburzając warunku (1). Z drugiej strony, koniec wystąpienia słowa $s_{i_2}$ musi następować za końcem wystąpienia $s_{i_1}$, jako że odwrotna sytuacja oznaczałaby, iż $s_{i_2}$ byłoby podsłowem $s_{i_1}$, co, jak wiemy, nie jest możliwe.

Konstruując słowo $t$, możemy więc albo ustawić $s_{i_2}$ tuż za $s_{i_1}$, albo próbować dopasować te słowa z nakładką, tzn. nałożyć pewien prefiks słowa $s_{i_2}$ na identyczny sufiks słowa $s_{i_1}$. Ponieważ słowo $t$ ma być najkrótsze możliwe, więc opłaca nam się wybrać nakładkę tak długą, jak tylko się da. Zauważmy, że to, jak długą nakładkę wybierzemy, nie ma żadnego znaczenia dla konfiguracji słów $s_{i_3}, \ldots, s_{i_m}$, ponieważ i tak słowo $s_{i_2}$ wystaje poza słowo $s_{i_1}$, a każde z tych kolejnych wystąpień następuje dalej niż $s_{i_2}$.

Zauważmy, że całe to rozumowanie można powtórzyć dla słów $s_{i_3}, \ldots, s_{i_m}$, otrzymując ostateczny wniosek, że słowo $t$ musi zaczynać się od $s_{i_1}$, kończyć się na $s_{i_m}$ oraz że należy zawsze wybierać najdłuższą możliwą nakładkę między $s_{i_{j-1}}$ a $s_{i_j}$. Spróbujmy zapisać ten wniosek bardziej formalnie.

Dla danych słów $u$ i $v$, *najdłuższą nakładką* tych słów (ang. *overlap*, oznaczenie: $ov(u,v)$) nazwiemy najdłuższe takie słowo $y$, że $u = xy$ i $v = yz$ dla pewnych niepustych słów $x$, $z$. W szczególności, słowo $y$ może być puste. Z kolei przez *prefiks* $pr(u,v)$ *słowa* $u$ *względem słowa* $v$ będziemy rozumieli słowo $x$ z definicji najdłuższej nakładki, tzn. $u = pr(u,v)ov(u,v)$. Dla przykładu,

$$ov(\texttt{aabababab, babbaa}) = \texttt{bab}, \quad pr(\texttt{aabababab, babbaa}) = \texttt{aaba},$$
$$ov(\texttt{abaab, abaab}) = \texttt{ab}, \quad pr(\texttt{abaab, abaab}) = \texttt{aba}.$$




Rys. 1: Struktura szukanego słowa $t$.

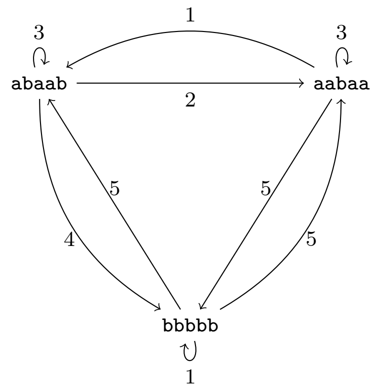
Korzystając z wprowadzonych właśnie oznaczeń, możemy zapisać słowo $t$ jako:

$$t \;=\; pr(s_{i_1}, s_{i_2}) \cdot pr(s_{i_2}, s_{i_3}) \cdot \ldots \cdot pr(s_{i_{m-1}}, s_{i_m}) \cdot s_{i_m}, \tag{3}$$

patrz także rys. 1. To oznacza, że gdybyśmy wiedzieli, jak dobrać kolejne słowa $s_{i_j}$, to umielibyśmy rozwiązać nasze zadanie. Tego jednak póki co nie wiemy, więc będziemy musieli jeszcze trochę pokombinować.

## Graf prefiksów

Okazuje się, że łatwiej rozwiązać nasze zadanie, jeśli dane zinterpretujemy jako graf. Dla danego zbioru $S$ skonstruujemy ważony, pełny graf skierowany $G = (V, E)$, który nazwiemy *grafem prefiksów*. Zbiór wierzchołków grafu $V = \{1, 2, \ldots, n\}$ utożsamiamy ze zbiorem słów $S$. Między każdą parą wierzchołków $(i, j)$ występuje krawędź skierowana, której wagę ustalamy jako $|pr(s_i, s_j)|$. Zauważmy, że w przypadku $i = j$ mamy w grafie pętlę, czyli formalnie $G$ jest multigrafem. Przykładowy graf prefiksów jest przedstawiony na rys. 2.




Rys. 2: Ilustracja grafu prefiksów dla zbioru słów $S = \{$`abaab`, `aabaa`, `bbbbb`$\}$.

Jeżeli przypomnimy sobie, co w poprzedniej sekcji udało nam się ustalić odnośnie do postaci wynikowego słowa $t$, to łatwo zauważymy, że graf prefiksów ma istotny związek z naszym wyjściowym problemem. Faktycznie, zapis (3) możemy odczytać następująco: *długość słowa $t$ jest równa długości najkrótszej ścieżki o $m$ wierzchołkach, łączącej pewne wierzchołki $i$, $j \in V$, powiększonej o $|s_j|$*. Słowo „najkrótsza” odnosi się tu, oczywiście, do sumy wag na krawędziach. Ta przyjemna własność sprowadza rozwiązanie naszego problemu do poszukiwania najkrótszych (w jakimś sensie) ścieżek w grafie prefiksów. Zanim zaczniemy kontynuować tę myśl, zastanówmy się jeszcze, w jaki sposób efektywnie wyznaczyć graf prefiksów dla danego zbioru $S$.

Do konstrukcji grafu $G$ wystarczą nam długości prefiksów postaci $|pr(s_i, s_j)|$. Będziemy je wyznaczać osobno dla każdej pary słów $(s_i, s_j)$. Przypomnijmy, że w tym celu wystarczy umieć obliczać najdłuższe nakładki postaci $|ov(s_i, s_j)|$ (czy Czytelnik pamięta dlaczego?). Najprostsze podejście, polegające na sprawdzaniu kolejnych możliwych długości nakładek, pozwala obliczyć każdą taką najdłuższą nakładkę w złożoności czasowej $O\left(\min(|s_i|, |s_j|)^2\right)$.

Aby poprawić ten wynik, zauważmy, że w przypadku, gdy $i \neq j$, $ov(s_i, s_j)$ jest równe długości najdłuższego prefiksu słowa $s_j$, który jest zarazem sufiksem słowa $s_i$, czyli jest równe długości najdłuższego właściwego *prefikso-sufiksu* słowa $s_j \# s_i$ (właściwego, czyli krótszego od całego tego słowa). Przy tym # jest symbolem, który (ewidentnie) nie należy do alfabetu angielskiego, a umieściliśmy go w środku po to, aby na pewno wynikowy prefikso-sufiks nie był dłuższy niż $s_i$ lub $s_j$. W przypadku szczególnym $i = j$ najdłuższa nakładka $s_i$ i $s_i$ odpowiada po prostu najdłuższemu właściwemu prefikso-sufiksowi samego słowa $s_i$.

Sprowadziliśmy zatem problem wyznaczania najdłuższych nakładek do obliczania najdłuższych właściwych prefikso-sufiksów pewnych słów. W tym miejscu warto przypomnieć, że długość najdłuższego prefikso-sufiksu dowolnego słowa można obliczyć w czasie proporcjonalnym do długości tego słowa, korzystając z funkcji prefiksowej z algorytmu Knutha-Morrisa-Pratta (patrz np. książki [19, 21]). To pokazuje, że $|ov(s_i, s_j)|$ możemy obliczyć (w każdym przypadku) w czasie $O(|s_i| + |s_j|)$, co daje następujący łączny koszt czasowy obliczenia wszystkich takich nakładek:

$$O\left(\sum_{i=1}^{n}\sum_{j=1}^{n}(|s_i|+|s_j|)\right) = O\left(\sum_{i=1}^{n}(n\cdot|s_i|+L)\right) = O\left(n\cdot\sum_{i=1}^{n}|s_i|+nL\right) = O(nL).$$

W ten właśnie sposób obliczano najdłuższe nakładki, a więc także i względne prefiksy, w rozwiązaniu wzorcowym. Warto w tym miejscu dodać, że istnieją inne metody wykonania tego podzadania. Jedna z nich polega na zastosowaniu drobnego oszustwa, a mianowicie haszowania: dla każdej pary $(s_i, s_j)$ przeglądamy kolejne możliwe długości nakładki i za pomocą haszy sprawdzamy w czasie stałym, czy odpowiedni prefiks i sufiks są równe. W ten sposób otrzymujemy inne rozwiązanie o złożoności czasowej $O(n\cdot L)$, które jest jednakże obarczone pewnym ryzykiem błędu. Więcej o metodzie haszowania można przeczytać w opracowaniu zadania *Antysymetria* w tej książeczce oraz w umieszczonych tam odnośnikach.

Na koniec dodajmy, że wszystkie nakładki można także obliczyć w optymalnej złożoności czasowej $O(n^2+L)$, co wymaga użycia bardziej skomplikowanych technik — zainteresowanych czytelników odsyłamy do artykułu [37], niestety dostępnego tylko w języku angielskim.

## Macierze

Sformułujmy pozostały do rozwiązania problem grafowy w sposób abstrakcyjny. Dany jest ważony, pełny graf skierowany $G=(V,E)$, $V=\{1,2,\ldots,n\}$, oraz liczba $m$. Dla każdej pary wierzchołków $(i,j)$ chcemy obliczyć długość najkrótszej ścieżki z $i$ do $j$ przechodzącej przez dokładnie $m-1$ krawędzi. Zauważmy, że rzeczywiście tyle nam wystarczy, aby rozwiązać wyjściowy problem.

Oznaczmy przez $T(m-1)$ poszukiwaną tabelkę odległości, rozmiaru $n\times n$. Dla utrudnienia, takie tabelki będziemy w dalszej części tekstu nazywać *macierzami*.

Macierz $T(0)$ reprezentuje najkrótsze możliwe ścieżki — jednowierzchołkowe — więc ma całkiem nieskomplikowaną postać: $T(0)_{i,i}=0$ dla każdego wierzchołka $i$, a poza tym $T(0)_{i,j}=\infty$ dla $i\neq j$, co wiąże się z tym, że takich ścieżek po prostu nie ma. Macierz $T(1)$ także wypełniamy całkiem łatwo — wystarczy w pole $T(1)_{i,j}$ wpisać wagę krawędzi prowadzącej z wierzchołka $i$ do wierzchołka $j$. Innymi słowy, $T(1)$ jest *macierzą sąsiedztwa* grafu $G$. Aby opisać to, co dzieje się dalej, przyjrzyjmy się ogólnej sytuacji: jak obliczyć $T(a+b)$ na podstawie $T(a)$ oraz $T(b)$? Odpowiedź na to pytanie daje następujący lemat.

**Lemat 1.** Dla dowolnych $a,b\geqslant 0$ oraz $i,j\in V$,

$$T(a+b)_{i,j} = \min\{T(a)_{i,k}+T(b)_{k,j} \;:\; k\in V\}.$$

**Dowód:** Żądaną równość łatwo uzasadnić, jeśli odpowiednio zinterpretuje się występujące w niej komórki macierzy. Zauważmy mianowicie, że ścieżkę reprezentującą $T(a+b)_{i,j}$ możemy podzielić na dwie części: pierwsze $a$ krawędzi i ostatnie $b$ krawędzi. Niech $k_0$ będzie wierzchołkiem, w którym spotykają się te części. Wówczas pierwsza część naszej ścieżki na pewno musi stanowić najkrótszą $a$-krawędziową ścieżkę z wierzchołka $i$ do wierzchołka $k_0$, podobnie druga część stanowi najkrótszą ścieżkę z $k_0$ do $j$. Stąd

$$T(a+b)_{i,j} = T(a)_{i,k_0} + T(b)_{k_0,j}.$$

Aby zakończyć dowód, wystarczy dodać, że dla wszystkich pozostałych wierzchołków $k$ musi zachodzić

$$T(a)_{i,k} + T(b)_{k,j} \geqslant T(a)_{i,k_0} + T(b)_{k_0,j},$$

gdyż w przeciwnym przypadku sklejając ścieżki odpowiadające $T(a)_{i,k}$ oraz $T(b)_{k,j}$, otrzymalibyśmy ścieżkę z $i$ do $j$ złożoną z $a+b$ krawędzi, ale krótszą niż $T(a+b)_{i,j}$, co nie jest możliwe. To pokazuje, że $T(a+b)_{i,j}$ jest równe owemu minimum, które występuje w tezie lematu. ∎

Zauważmy teraz, że jeśli dla dowolnych macierzy $M$, $N$ rozmiaru $n \times n$ zdefiniujemy działanie $\oplus$ jako[1]:

$$(M \oplus N)_{i,j} \stackrel{\text{def}}{=} \min\{M_{i,k} + N_{k,j} \;:\; k = 1, 2, \ldots, n\},$$

to wzór zawarty w Lemacie 1 możemy zapisać w bardzo prostej postaci:

$$T(a+b) = T(a) \oplus T(b) \quad \text{dla } a, b \geqslant 0. \tag{4}$$

Korzystając z powyższego wzoru, bardzo łatwo skonstruować *jakieś* rozwiązanie naszego zadania: aby obliczyć $T(m-1)$, wystarczy zastosować tenże wzór $m-2$ razy, za każdym razem podstawiając $a = 1$. Przypomnijmy jednak, że parametr $m$ może być naprawdę duży, więc takim rozwiązaniem nie możemy się jeszcze zadowolić.

Okazuje się, że wzór (4) stanowi klucz również do bardziej efektywnej metody obliczania $T(m-1)$. Możemy mianowicie zastosować postępowanie analogiczne do szybkiego potęgowania binarnego[2], czyli korzystające z następujących wzorów rekurencyjnych:

$$T(2k) = T(k) \oplus T(k), \qquad T(2k+1) = T(1) \oplus T(2k).$$

W ten sposób możemy obliczyć $T(m-1)$, wykonując działanie $\oplus$ zaledwie $O(\log m)$ razy. Dla macierzy $n \times n$ koszt czasowy wykonania działania $\oplus$ to $O(n^3)$, ponieważ musimy obliczyć $n^2$ wartości, z których każda to minimum z $n$ sum par liczb. To oznacza, że umiemy obliczyć $T(m-1)$ w złożoności czasowej $O(n^3 \log m)$.

## Rozwiązanie wzorcowe

W rozwiązaniu wzorcowym najpierw, w czasie $O(n \cdot L)$, obliczamy macierz sąsiedztwa grafu prefiksów $G$, a następnie podstawiamy ją za $T(1)$ i obliczamy $T(m-1)$ za

[1] Czytelnicy zaznajomieni z mnożeniem macierzy mogą zauważyć podobieństwo działania $\oplus$ do mnożenia macierzy — gdyby operację „min” zastąpić przez sumę, a dodawanie przez mnożenie, to otrzymalibyśmy dokładnie standardowe mnożenie macierzy.

[2] Można pokazać, że działanie $\oplus$ (nazywane z ang. *min-plus product*) jest łączne i w związku z tym macierz $T(m-1)$ można tak naprawdę zapisać jako $T(1)^{m-1}$, czyli $(m-1)$-szą potęgę $T(1)$ względem działania $\oplus$.

pomocą opisanej metody binarnej, w czasie $O(n^3 \log m)$. Korzystając z komórek tej macierzy, obliczamy wynik ze wzoru:

$$\min\left\{T(m-1)_{i,j} + |s_j| \;:\; i,j = 1,2,\ldots,n\right\}.$$

Złożoność czasowa rozwiązania wzorcowego to $O(n \cdot (n^2 \log m + L))$, a pamięciowa $O(n^2 + L)$. Jego implementacje można znaleźć w plikach `cho.cpp`, `cho1.pas` oraz `cho2.cpp`.

## Inne rozwiązania

W tym zadaniu rozwiązania powolne to przede wszystkim nieefektywne implementacje pomysłów zawartych w rozwiązaniu wzorcowym. Wyróżniamy tu rozwiązania obliczające graf prefiksów metodą siłową — złożoność czasowa $O(L^2 + n^3 \log m)$, implementacje w plikach `chos1.cpp` i `chos2.pas`, rozwiązania obliczające $T(m-1)$ za pomocą $(m-2)$-krotnego wykonywania działania $\oplus$ — złożoność czasowa $O(n \cdot L + n^3 \cdot m)$, implementacje w plikach `chos3.cpp` i `chos4.pas`, oraz rozwiązania mające obie te wady — złożoność czasowa $O(L^2 + n^3 \cdot m)$, pliki `chos5.cpp` i `chos6.pas`. Rozwiązania pierwszego typu otrzymywały na zawodach około 80 punktów, natomiast pozostałe od około 30 do 50 punktów.

Wśród rozwiązań niepoprawnych warto zwrócić uwagę na różne heurystyki oparte na konstruowaniu optymalnej ścieżki jedynie za pomocą pojedynczych cykli w grafie $G$ (pliki `chob1.cpp`, `chob2.cpp`, `chob5.cpp`, zdobywały do 40 punktów) oraz implementacje rozwiązania wzorcowego nieużywające zmiennych całkowitych 64-bitowych (`chob3.cpp`, zdobywało 80 punktów).

## Testy

Do sprawdzenia rozwiązań użyto 10 grup testów. Poza specyficznymi testami *1d*, *1e*, *6d* i *8d*, poszczególne testy w grupach były następujących typów:

typ a: testy losowe,

typ b: testy, w których kolejne imiona chomików konstruowano, wybierając losowy sufiks poprzedniego imienia i używając go jako prefiksu nowego imienia (np. `abcd`, `bcdefgh`, `ghi`),

typ c: testy podobne do testów typu *b*, w których dodatkowo do wybranych imion doklejano losowe sufiksy, będące prefiksami już skonstruowanych imion, tworząc pewnego rodzaju cykle (np. `abcd`, `cdefgh`, `fghabc`),

typ d: testy wydajnościowe, w których imiona chomików mają specjalną strukturę, wymuszającą wykorzystanie efektywnych metod do obliczania najdłuższych nakładek.

Na następnej stronie znajduje się tabela z podstawowymi statystykami wykorzystanych testów ($n$ — liczba imion chomików, $L$ — suma długości imion, $m$ — dolne ograniczenie na liczbę wystąpień imion chomików w wynikowym słowie).

| **Nazwa** | **n** | **L** | **m** |
|---|---|---|---|
| *cho1a.in* | 7 | 134 | 45 |
| *cho1b.in* | 9 | 126 | 32 |
| *cho1c.in* | 7 | 100 | 37 |
| *cho1d.in* | 2 | 4 | 1 |
| *cho1e.in* | 8 | 64 | 50 |
| *cho2a.in* | 16 | 452 | 87 |
| *cho2b.in* | 17 | 536 | 78 |
| *cho2c.in* | 24 | 596 | 86 |
| *cho3a.in* | 42 | 464 | 650 |
| *cho3b.in* | 35 | 508 | 744 |
| *cho3c.in* | 47 | 873 | 994 |
| *cho4a.in* | 61 | 2 394 | 5 071 |
| *cho4b.in* | 94 | 2 652 | 4 781 |
| *cho4c.in* | 55 | 3 005 | 4 016 |
| *cho5a.in* | 76 | 3 652 | 38 980 |
| *cho5b.in* | 73 | 3 946 | 32 521 |
| *cho5c.in* | 79 | 5 780 | 30 944 |
| *cho6a.in* | 90 | 6 664 | 160 907 |
| *cho6b.in* | 85 | 8 258 | 174 302 |
| *cho6c.in* | 104 | 7 152 | 168 395 |
| *cho6d.in* | 101 | 10 201 | 524 288 |

| **Nazwa** | **n** | **L** | **m** |
|---|---|---|---|
| *cho7a.in* | 94 | 13 706 | 795 662 |
| *cho7b.in* | 93 | 18 636 | 985 466 |
| *cho7c.in* | 113 | 13 960 | 806 575 |
| *cho8a.in* | 134 | 16 124 | 9 161 573 |
| *cho8b.in* | 155 | 25 282 | 8 508 759 |
| *cho8c.in* | 92 | 24 784 | 9 180 994 |
| *cho8d.in* | 141 | 19 881 | 7 654 210 |
| *cho9a.in* | 104 | 28 488 | 52 909 188 |
| *cho9b.in* | 114 | 37 786 | 98 565 618 |
| *cho9c.in* | 98 | 37 144 | 67 504 299 |
| *cho9d.in* | 100 | 59 850 | 717 226 983 |
| *cho10a.in* | 7 | 95 991 | 889 592 424 |
| *cho10b.in* | 187 | 54 081 | 523 267 787 |
| *cho10c.in* | 185 | 64 190 | 925 285 703 |
| *cho10d.in* | 100 | 99 850 | 811 650 835 |