Zepsuta klawiatura (rozwiązanie)

Autor zadania: **Lech Duraj**
Opracowanie: **Jakub Misiaszek, Paweł Zalewski**
Opis rozwiązania: **Bartosz Kostka**

Olimpiada Informatyczna Juniorów

Zadanie

Na wejściu otrzymujemy ciąg znaków S składający się z małych liter alfabetu angielskiego. Naszym zadaniem jest przetworzenie tego ciągu, symulując działanie specjalnego klawisza `d`, którego naciśnięcie powoduje usunięcie ostatniego wystąpienia największej (w porządku alfabetycznym) litery w dotychczas napisanym tekście. Jeśli tekst jest pusty, `d` nic nie robi.

Rozwiązanie brutalne — $O(|S|^2)$

Najprostszym podejściem jest bezpośrednia symulacja procesu opisanego w zadaniu. Przetwarzamy kolejne znaki wejściowego ciągu S i na bieżąco budujemy wynikowy napis, np. w zmiennej typu `string` w C++ lub na liście znaków w Pythonie[1].

- Jeśli napotkany znak jest literą inną niż `d`, dopisujemy go na koniec wynikowego napisu.
- Jeśli napotkany znak to `d`, musimy znaleźć i usunąć odpowiednią literę z wynikowego napisu. W tym celu przeszukujemy cały aktualny napis od końca, aby znaleźć pozycję ostatniego wystąpienia największej leksykograficznie litery. Po znalezieniu usuwamy znak na tej pozycji.

Operacja wyszukiwania znaku do usunięcia oraz sama operacja usunięcia z ciągu znaków (np. `string::erase` w C++ lub `list.pop()` w Pythonie, jeśli napis przechowujemy w liście) zajmują czas proporcjonalny do długości aktualnego napisu. Ponieważ napis może mieć długość do $|S|$, a operację usuwania możemy wykonać do $|S|$ razy, całkowita złożoność czasowa tego rozwiązania wynosi $O(|S|^2)$. Przy $|S|$ do 1 000 000 jest to rozwiązanie zbyt wolne, ale pozwala na zdobycie 25 punktów.

zkl_brut.py

```
S = input()

obecnie = []
for c in S:
    if c != 'd':
        obecnie.append(c)
    elif obecnie:
        pos = len(obecnie) - 1
        for i in range(len(obecnie) - 1, -1, -1):
            if obecnie[i] > obecnie[pos]:
                pos = i
        obecnie.pop(pos)

print("".join(obecnie))
```

[1] Zwróć uwagę, że dodawanie znaków do zmiennej type `str` w Pythonie jest liniowe względem długości całego napisu, zatem dodawanie znak po znaku powoduje, że złożoność tego kroku jest kwadratowa.

Olimpiada finansowana jest ze środków Ministerstwa Edukacji Narodowej w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2025/2026, 2026/2027, 2027/2028".

zkl_brut.cpp

```cpp
#include <iostream>
#include <string>

using namespace std;

int main() {
  string S;
  cin >> S;

  string obecnie = "";
  for (char c : S) {
    if (c != 'd') {
      obecnie += c;
    } else if (!obecnie.empty()) {
      int pos = (int)obecnie.size() - 1;
      for (int i = pos; i >= 0; i--) {
        if (obecnie[i] > obecnie[pos]) {
          pos = i;
        }
      }
      obecnie.erase(pos, 1);
    }
  }
  cout << obecnie << "\n";
}
```

Rozwiązanie wzorcowe — $O(|S| \cdot |\Sigma|)$

Aby przyśpieszyć rozwiązanie, musimy zoptymalizować operację znajdowania znaku do usunięcia. Zamiast za każdym razem przeszukiwać cały napis, możemy przechowywać pozycje (indeksy) wystąpień poszczególnych liter w pomocniczej strukturze. Idealnie nadaje się do tego tablica 26 wektorów (lub stosów), gdzie każdy element tablicy odpowiada jednej literze alfabetu.

Przechodzimy przez wejściowy ciąg S znak po znaku, od lewej do prawej.

- Gdy napotykamy literę c (różną od `d`) na pozycji i, dodajemy indeks i do wektora dla litery c.
- Gdy napotykamy znak `d`, przeszukujemy naszą tablicę wektorów, zaczynając od litery `z` w dół do `a`. Pierwszy napotkany niepusty wektor odpowiada największej leksykograficznie literze obecnej w napisie. Usuwamy z końca tego wektora ostatni zapisany indeks.

Po przetworzeniu całego ciągu S wektory zawierają indeksy liter, które pozostały w finalnym napisie. Aby odtworzyć wynik, nie możemy po prostu usuwać znaków z oryginalnego ciągu w trakcie jego przetwarzania, gdyż byłoby to nieefektywne. Zamiast tego, możemy użyć dodatkowej tablicy (np. typu `bool`) o rozmiarze $|S|$, w której będziemy oznaczać usunięte znaki. W rozwiązaniach przytoczonych poniżej zamiast tablicy `bool`, oryginalny ciąg jest modyfikowany przez wstawienie specjalnego znaku (#) w miejsce usuniętych liter.

Na koniec wystarczy przejść po zmodyfikowanym ciągu i złączyć wszystkie znaki, które nie zostały usunięte.

Dla lepszego zobrazowania tego rozwiązania, rozważmy napis `eefeddebd`. Będziemy opisywali akcje po przetwarzaniu kolejnych liter.

- `e` (indeks 0): dla stosu dla litery 'e' dodajemy indeks 0: `[0]`
- `e` (indeks 1): dodajemy indeks 1 do stosu dla litery 'e': `[0, 1]`
- `f` (indeks 2): dodajemy indeks 2 do stosu dla litery 'f': `[2]`
- `e` (indeks 3): stos dla 'e': `[0, 1, 3]`

Olimpiada finansowana jest ze środków Ministerstwa Edukacji Narodowej w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2025/2026, 2026/2027, 2027/2028".

- `d`: Natrafiamy na specjalną literę `d`. Przeglądamy stosy od największej litery do najmniejszej, szukając niepustego stosu. W tym przypadku pierwszym niepustym stosem jest stos dla 'f'. Usuwamy zatem ostatnie wystąpienie z tego stosu (jest to `f` z ind. 2). Stos dla 'f' jest teraz pusty.
- `d`: Usuwamy ostatnie wystąpienie największej litery (`e` z ind. 3). Stos dla 'e' to `[0, 1]`.
- `e` (indeks 6): stos dla 'e': `[0, 1, 6]`
- `b` (indeks 7): stos dla 'b': `[7]`
- `d`: Usuwamy ostatnie wystąpienie największej litery (`e` z ind. 6). Stos dla 'e' to `[0, 1]`.

Po przetworzeniu całego ciągu, w strukturach danych pozostają indeksy: dla 'e' — `[0, 1]`, dla 'b' — `[7]`. Po posortowaniu wszystkich pozostałych indeksów i odczytaniu znaków z oryginalnego napisu otrzymujemy `eeb`.

Złożoność czasowa tej metody to $O(|S| \cdot |\Sigma|)$, gdzie $|\Sigma|$ to rozmiar alfabetu (26). Dla każdego znaku `d` w najgorszym przypadku musimy przejrzeć całą tablicę wektorów. Ponieważ $|\Sigma|$ jest małą stałą, złożoność jest w praktyce liniowa. Złożoność pamięciowa to $O(|S|)$ na przechowywanie indeksów.

zkl.cpp

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

const int rozmiar_alfabetu = 26; // ewentualnie 'z' - 'a' + 1
vector<int> pozycje[rozmiar_alfabetu];

int main() {
  string S;
  cin >> S;

  for (int i = 0; i < (int)S.size(); i++) {
    if (S[i] != 'd') {
      pozycje[S[i] - 'a'].push_back(i);
    } else {
      // Zamiast 'd' będziemy trzymali '#'.
      S[i] = '#';
      // Iterujemy od 'z' do 'a', aby znaleźć największą literę.
      for (char litera = 'z'; litera >= 'a'; litera--) {
        int pozycja_litery = litera - 'a';
        if (!pozycje[pozycja_litery].empty()) {
          // Usunięte litery także oznaczymy jako '#'.
          S[pozycje[pozycja_litery].back()] = '#';
          pozycje[pozycja_litery].pop_back();
          break;
        }
      }
    }
  }
  string wynik = "";
  for (char c : S) {
    if (c != '#') {
      wynik.push_back(c);
    }
  }
  cout << wynik << "\n";
}
```


Olimpiada finansowana jest ze środków Ministerstwa Edukacji Narodowej w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2025/2026, 2026/2027, 2027/2028".

zkl.py

```python
def main():
    S = list(input())

    rozmiar_alfabetu = 26
    pozycje = [[] for _ in range(rozmiar_alfabetu)]

    for i, char in enumerate(S):
        if char != 'd':
            pozycje[ord(char) - ord('a')].append(i)
        else:
            # Zamiast 'd' będziemy trzymali '#'.
            S[i] = '#'
            # Iterujemy od 'z' do 'a', aby znaleźć największą literę.
            for litera_ord in range(ord('z'), ord('a') - 1, -1):
                pozycja_litery = litera_ord - ord('a')
                if pozycje[pozycja_litery]:
                    # Usunięte litery także oznaczamy jako '#'.
                    # .pop() zwraca i usuwa ostatni element.
                    S[pozycje[pozycja_litery].pop()] = '#'
                    break

    wynik = []
    for c in S:
        if c != '#':
            wynik.append(c)

    print("".join(wynik))


if __name__ == "__main__":
    main()
```

Rozwiązanie alternatywne — $O(|S| \log |S|)$

Innym podejściem jest użycie kolejki priorytetowej. Przetwarzamy ciąg wejściowy i dla każdej napotkanej litery c na pozycji i, wstawiamy do kolejki parę $\{c, i\}$. Kolejka priorytetowa (skonfigurowana jako max-heap) automatycznie utrzyma na wierzchołku element z największą literą (a w przypadku remisu, z największym indeksem). Gdy napotkamy znak d, po prostu usuwamy element z wierzchołka kolejki.

Po przetworzeniu całego ciągu S, w kolejce pozostaną pary $\{liter, indeks\}$ odpowiadające znakom w końcowym tekście. Aby je wypisać w poprawnej kolejności, musimy je wszystkie wyjąć z kolejki, umieścić w wektorze, a następnie posortować według indeksów.

Złożoność czasowa tego rozwiązania to $O(|S| \log |S|)$ ze względu na operacje na kolejce priorytetowej oraz końcowe sortowanie. Jest ono nieco wolniejsze od wzorcówki, ale również może uzyskać pełną punktację.

zkl_log.cpp

```cpp
#include <iostream>
#include <algorithm>
#include <string>
#include <queue>
#include <vector>
#include <utility>

using namespace std;

int main() {
```


Olimpiada finansowana jest ze środków Ministerstwa Edukacji Narodowej w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2025/2026, 2026/2027, 2027/2028".

```cpp
    string S;
    cin >> S;

    // W kolejce priorytetowej przechowujemy pary znak, pozycja.
    // Domyślnie sortuje ona malejąco, więc na wierzchołku zawsze będzie
    // największy leksykograficznie znak.
    priority_queue<pair<char, int>> kolejka;
    for (int i = 0; i < (int) S.size(); i++) {
        if (S[i] != 'd') {
            kolejka.push({S[i], i});
        }
        else if (!kolejka.empty()) {
            // Wciśnięcie 'd' usuwa największy element.
            kolejka.pop();
        }
    }

    // Aby odtworzyć wynik, musimy posortować pozostałe znaki po ich
    // oryginalnych pozycjach.  Zbieramy pozostałe pary do wektora, odwracając je.
    vector<pair<int, char>> pozostale_znaki;
    while (!kolejka.empty()) {
        auto [c, poz] = kolejka.top();
        kolejka.pop();
        pozostale_znaki.push_back({poz, c});
    }

    // Sortujemy wektor po pozycjach.
    sort(pozostale_znaki.begin(), pozostale_znaki.end());

    string wynik = "";
    for (auto [poz, c] : pozostale_znaki) {
        wynik += c;
    }
    cout << wynik << "\n";
}
```

zkl_log.py

```python
import heapq


def main():
    S = input()

    # W kolejce priorytetowej (min-heap) przechowujemy pary -ord(znak), -pozycja.
    # Negacja wartości pozwala symulować max-heap, który na wierzchołku
    # utrzyma największy leksykograficznie znak z największą pozycją.
    kolejka = []
    for i, c in enumerate(S):
        if c != 'd':
            heapq.heappush(kolejka, (-ord(c), -i))
        elif kolejka:
            # Wciśnięcie 'd' usuwa największy element.
            heapq.heappop(kolejka)

    # Aby odtworzyć wynik, musimy posortować pozostałe
    # znaki po ich oryginalnych pozycjach.
    # Zbieramy pozostałe pary do listy, odwracając je.
    pozostale_znaki = []
    while kolejka:
```


Olimpiada finansowana jest ze środków Ministerstwa Edukacji Narodowej w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2025/2026, 2026/2027, 2027/2028".

```
23          neg_ord_c, neg_i = heapq.heappop(kolejka)
24          # Odwracamy parę do postaci (pozycja, znak)
25          pozostale_znaki.append((-neg_i, chr(-neg_ord_c)))
26
27      # Sortujemy wektor po pozycjach.
28      pozostale_znaki.sort()
29
30      wynik = []
31      for _, c in pozostale_znaki:
32          wynik.append(c)
33
34      print("".join(wynik))
35
36
37  if __name__ == "__main__":
38      main()
```


Olimpiada finansowana jest ze środków Ministerstwa Edukacji Narodowej w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2025/2026, 2026/2027, 2027/2028".