Sortowanie pozycyjne słów w języku Python

Źródło: Markus Spiske, domena publiczna.

Z sortowaniem słów mamy do czynienia chociażby w słowniku, encyklopedii czy dzienniku szkolnym. W e-materiale Sortowanie pozycyjne słów dowiedzieliśmy się, jak do porządkowania słów użyć algorytmu sortowania pozycyjnego.

W tym e-materiale zaimplementujemy ten algorytm w języku Python.

Ciekawi cię, jak wyglądają implementacje w innych językach programowania? Możesz się z nimi zapoznać w dwóch pozostałych e-materiałach z serii:

- Sortowanie pozycyjne słów w języku C++,
- Sortowanie pozycyjne słów w języku Java.

Więcej zadań? Sięgnij do: Sortowanie pozycyjne słów – zadania maturalne.

> **Twoje cele**
>
> - Przeanalizujesz algorytm sortowania pozycyjnego słów, zaimplementowany w języku Python.
> - Napiszesz w języku Python program sortujący listę imion, wykorzystując algorytm sortowania pozycyjnego.
> - Zaimplementujesz algorytm wizualizujący liczbę operacji niezbędnych do wykonania sortowania pozycyjnego słów.

Film samouczek

W życiu codziennym często mamy do czynienia ze zbiorami posortowanych słów. Są to wszelkiego rodzaju katalogi czy listy, uporządkowane najczęściej zgodnie z kryterium alfabetycznym. Jak już wiesz, sortowanie pozycyjne słów (*radix sort*) służy do sortowania zbiorów danych według porządku leksykograficznego. Wymaga ono zastosowania algorytmu pomocniczego, który musi być stabilny. W tej sekcji posłużymy się sortowaniem przez zliczanie.

Polecenie 1

Napisz program sortujący niemalejąco listę imion, wykorzystując sortowanie pozycyjne słów (*radix sort*). Jako algorytm pomocniczy wykorzystaj sortowanie przez zliczanie (*counting sort*).

Specyfikacja problemu:

Dane:

- *dane* – jednowymiarowa lista przechowująca łańcuchy znaków, która zawiera imiona do posortowania

Wynik:

- program wypisuje posortowane w kolejności niemalejącej imiona z listy `dane`, wykorzystując sortowanie pozycyjne słów; kolejne elementy powinny być oddzielone pojedynczym znakiem spacji

Twoje zadania

1. Posortuj niemalejąco listę `dane` algorytmem *radix sort*. Wypisz elementy posortowanej listy oddzielone pojedynczym znakiem spacji.

```
dane = ["igor", "ala", "adam"]

# uzupełnij kod
```

```
```

Polecenie 2

Porównaj swoje rozwiązanie z przedstawionym w filmie.

Film dostępny pod adresem /preview/resource/RsEQWlpwGGZNH

Film przedstawia sortowanie pozycyjne słów w języku Python.

Kod programu zaprezentowanego w filmie:

Plik o rozmiarze 1.77 KB w języku polskim

Ważne!

Rozwiązanie zadania przedstawionego w filmie znajdziesz w sekcji „Przeczytaj”.

Polecenie 3

Dodaj do swojego programu komentarze tak, żeby był zrozumiały dla osoby, która nie potrafi programować.

Polecenie 4

Zapisz implementację programu z **Polecenia 1** wykorzystującą inny stabilny algorytm sortowania.

Przeczytaj

Implementacja algorytmu sortowania pozycyjnego słów w języku Python

Algorytm pomocniczy – sortowanie przez zliczanie

Sortowanie pozycyjne służy do sortowania zbiorów danych według porządku leksykograficznego. Algorytm ten wewnętrznie używa pomocniczego – innego niż sortowanie pozycyjne – algorytmu sortowania, który jest odpowiedzialny za sortowanie według poszczególnych elementów w ciągach.

Pamiętajmy, że zastosowany algorytm musi być stabilny – możemy użyć dowolnego algorytmu, jeśli tylko spełnia on ten warunek. Sortowanie pozycyjne jest często łączone np. z sortowaniem przez zliczanie, ponieważ oba algorytmy są relatywnie szybkie dla rozważanego typu danych (ciągów znaków).

Algorytm sortowania przez zliczanie został wykorzystany jako algorytm pomocniczy w sekcji „Film samouczek". Przypomnijmy jego implementację.

Specyfikacja problemu:

Dane:

- `dane` – jednowymiarowa lista przechowująca łańcuchy znaków, która zawiera imiona do posortowania

Wynik:

- program wypisuje posortowane w kolejności niemalejącej imiona z listy `dane`, wykorzystując sortowanie pozycyjne słów; kolejne elementy powinny być oddzielone pojedynczym znakiem spacji

Przetestujemy działanie programu dla następującej listy `dane = [ "igor", "ala", "adam" ]`.

Kod zawiera trzy funkcje realizujące algorytm.

Zadaniem funkcji `przygotuj_napisy()` jest przygotowanie ciągów w liście `dane` tak, aby wszystkie miały taką samą długość równą wartości zmiennej `dlugosc_najdluzszego_slowa`. Jeżeli któryś z ciągów jest krótszy, zostaje na końcu (po ostatnim znaku) uzupełniony znakami grawisu – ` (jest to ostatni – przed małymi literami – znak w tablicy ASCII).

Funkcja `wyczysc_napisy()` ma zadanie odwrotne – po wykonaniu sortowania musi usunąć nadmiarowe znaki grawisów, zastępując je spacjami.

Funkcja `sortowanie_przez_zliczanie()` realizuje algorytm sortowania przez zliczanie.

```
def przygotuj_napisy(dane, dlugosc_najdluzszego_slowa):
    for indeks, slowo in enumerate(dane):
        while len(slowo) < dlugosc_najdluzszego_slowa:
            slowo += "`"
        dane[indeks] = slowo

def wyczysc_napisy(dane):
    for indeks, slowo in enumerate(dane):
        dane[indeks] = slowo.replace("`", "")

def sortowanie_pozycyjne_slow(dane):
    dlugosc_najdluzszego_slowa = max([len(slowo) for slowo in dan

    przygotuj_napisy(dane, dlugosc_najdluzszego_slowa)

    for i in range(dlugosc_najdluzszego_slowa - 1, -1, -1):
        sortowanie_przez_zliczanie(dane, i)

    wyczysc_napisy(dane)

    return dane


def sortowanie_przez_zliczanie(dane, indeks_znaku):
    lista_zliczen_liczb = [0 for x in range(27)]
    for slowo in dane:
        odczytany_kod_znaku = ord(slowo[indeks_znaku]) - ord("`")
        lista_zliczen_liczb[odczytany_kod_znaku] += 1

    for i in range(1, len(lista_zliczen_liczb)):
        lista_zliczen_liczb[i] += lista_zliczen_liczb[i - 1]

    temp = ["" for slowo in dane]

    for i in range(len(dane) - 1, -1, -1):
```

```
        odczytany_kod_znaku = ord(dane[i][indeks_znaku]) - ord("`
        indeks_w_liscie_wynikowej = lista_zliczen_liczb[odczytany
        temp[indeks_w_liscie_wynikowej] = dane[i]
        lista_zliczen_liczb[odczytany_kod_znaku] -= 1

    for i in range(len(temp)):
        dane[i] = temp[i]

dane = [ "igor", "ala", "adam" ]
dane = sortowanie_pozycyjne_slow(dane)

for i in range(0, len(dane)):
    print(dane[i])
```

Efektem działania programu jest wyświetlenie danych ciągów znaków już po posortowaniu. Ciągi wyświetlane są w kolejnych wierszach:

```
adam
ala
igor
```

Algorytm pomocniczy – sortowanie bąbelkowe

Zapoznajmy się z implementacją wykorzystującą inny algorytm pomocniczy – algorytm sortowania bąbelkowego.

> **Przykład 1**
>
> Uporządkujmy listę imion w kolejności alfabetycznej. Wszystkie te imiona składają się z czterech liter, dlatego wykonamy cztery iteracje – w każdej będziemy porządkować odpowiednie pozycje słów. W każdej z iteracji część kubełków pozostaje pusta. Liczba kubełków jest zdeterminowana liczbą różnych liter występujących w imionach.

```
# wejściowa lista imion
"KUBA",
"EMIL",
"ADAM",
"IGOR",
"ALEK",
"HUGO",
"ERYK",
```

```
9
10 # lista kubełków
11 'K', 'U', 'B', 'A', 'E', 'M', 'I', 'L', 'D', 'G', 'O', 'R', 'H'
```

W pierwszej iteracji sprawdzamy ostatnią literę:

```
1 # wypisane będą tylko kubełki "zapełnione"
2 'A' - "KUBA"
3 'K' - "ALEK", "ERYK"
4 'L' - "EMIL"
5 'M' - "ADAM"
6 'O' - "HUGO"
7 'R' - "IGOR"
```

W drugiej iteracji sprawdzamy przedostatnią literę:

```
1 # wypisane będą tylko kubełki "zapełnione"
2 'A' - "ADAM"
3 'B' - "KUBA"
4 'E' - "ALEK"
5 'G' - "HUGO"
6 'I' - "EMIL"
7 'O' - "IGOR"
8 'Y' - "ERYK"
```

W trzeciej iteracji sprawdzamy literę trzecią od końca:

```
1 # wypisane będą tylko kubełki "zapełnione"
2 'D' - "ADAM:"
3 'G' - "IGOR"
4 'L' - "ALEK"
5 'M' - "EMIL"
6 'R' - "ERYK"
7 'U' - "KUBA", "HUGO"
```

W czwartej iteracji sprawdzamy literę czwartą od końca (czyli w tym przypadku pierwszą):

```
# wypisane będą tylko kubełki "zapełnione"
'A' - "ADAM", "ALEK",
'E' - "EMIL", "ERYK"
'H' - "HUGO"
'I' - "IGOR"
'K' - "KUBA"
```

W efekcie uzyskujemy imiona posortowane alfabetycznie:

```
"ADAM", "ALEK", "EMIL", "ERYK", "HUGO", "IGOR", "KUBA"
```

Ważne!

Możemy z łatwością zauważyć, jak w kolejnych kubełkach pojawiają się słowa. Problem pojawia się w przypadku słów o różnej długości – wówczas musimy zadecydować, gdzie umieścić słowa krótsze. Kiedy wykraczamy poza ostatnią literę takiego słowa, dopisujemy słowo do tworzonego w danej iteracji ciągu wszystkich słów, do którego po zakończeniu wypełniania kubełków będziemy dopisywać słowa z opróżnianych kubełków.

Przykład 2

Przygotujmy program, który zrealizuje sortowanie pozycyjne słów.

Specyfikacja problemu

Dane:

- `dane` – lista ciągów znaków zawierająca teksty do posortowania

Wynik:

- `lista_posortowana` – posortowana lista ciągów znaków

Działanie programu przetestujemy dla następującej listy:

```
dane = [ "igor", "ala", "adam" ]
```

Widzimy, że liczba znaków w danych wejściowych może być różna, więc musimy napisać funkcję pomocniczą, która sprawdzi, ile liter ma najdłuższe słowo. Następnie funkcja zwróci liczbę iteracji oraz stworzy listę unikatowych liter występujących w słowach. Celem jest przygotowanie słownika, który będziemy wykorzystywać w procesie sortowania bąbelkowego.

Implementację algorytmu sortowania bąbelkowego znajdziesz w e-materiale mu poświęconym Sortowanie bąbelkowe w języku Python.

```
def przygotuj_napisy(dane, dlugosc_najdluzszego_slowa):
    for indeks, slowo in enumerate(dane):
        while len(slowo) < dlugosc_najdluzszego_slowa:
            slowo += "`"
        dane[indeks] = slowo

def wyczysc_napisy(dane):
    for indeks, slowo in enumerate(dane):
        dane[indeks] = slowo.replace("`", "")

def sortowanie_pozycyjne_slow(dane):
    dlugosc_najdluzszego_slowa = max([len(slowo) for slowo in d

    przygotuj_napisy(dane, dlugosc_najdluzszego_slowa)

    for i in range(dlugosc_najdluzszego_slowa - 1, -1, -1):
        sortowanie_babelkowe(dane)

    wyczysc_napisy(dane)

    return dane


def sortowanie_babelkowe(dane):
    for i in range(0, n - 1):
        wykonanie = False
        for j in range(0, n - i - 1):
            if dane[j] > dane[j + 1]:
                wykonanie = True
                pomoc = dane[j + 1]
                dane[j + 1] = dane[j]
                dane[j] = pomoc
        if not wykonanie:
            break

dane = [ "igor", "ala", "adam" ]
n = len(dane)
dane = sortowanie_pozycyjne_slow(dane)
```

```
39
40 for i in range(0, len(dane)):
41     print(dane[i])
```

Wynikiem działania jest posortowana lista: `["adam","ala","igor"]`

Ciekawostka

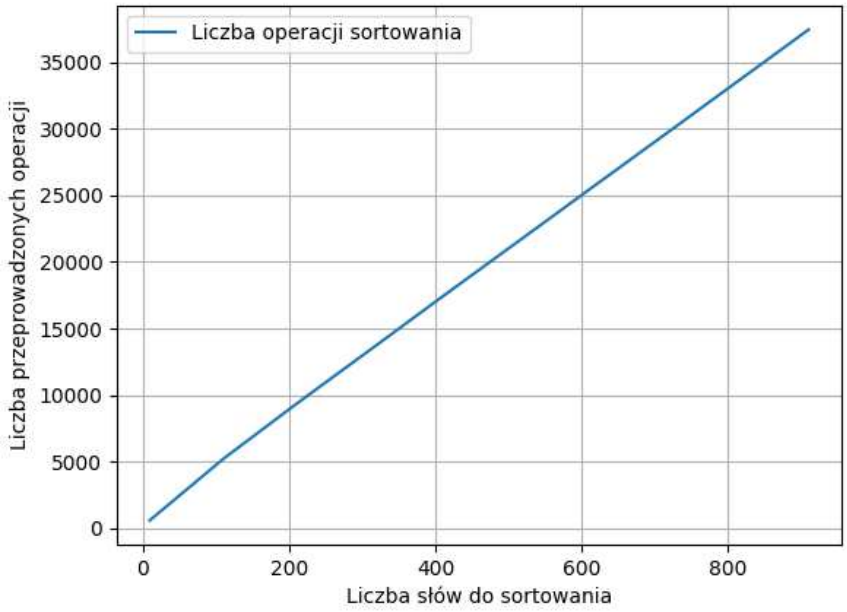
Korzystając z tej funkcji, stworzymy program wizualizujący – za pomocą modułu matplotlib – zależność operacji sortowania losowej listy słów od liczby elementów wejściowej listy danych oraz od długości największego elementu listy. Aby wygenerować różne słowa, użyjemy losowych liter alfabetu łacińskiego. Operacjami dominującymi w tym algorytmie są operacje umieszczania elementów w kubełku.

```
1  from random import randint, choice
2  import matplotlib.pyplot as plt
3
4  def sortowanie_poz_slow_wizualizacja(lista_slow):
5
6      def kubelki_iteracje(lista):
7          kub = []
8          dl = 0
9          for w in lista:
10             if len(w) > dl:
11                 dl = len(w)
12             for l in w:
13                 if not l.upper() in kub:
14                     kub.append(l.upper())
15         return (sorted(kub), dl)
16
17     def sortuj(lista, pozycja, kb):
18         oper = 0
19         l = []
20         kubelki = { p : [] for p in kb}
21         for w in lista:
22             oper += 1
23             if len(w) < pozycja:
24                 l.insert(0, w)
25             else:
26                 kubelki[w[pozycja-1].upper()].append(w)
27         # przepisanie z kubełków do listy
28         for i in kubelki:
```

```
            if len(kubelki[i]) > 0:
                l += kubelki[i]
                oper += 1
        # zwracamy posortowane wg określonej pozycji
        return l[:], oper


    kb, il = kubelki_iteracje(lista_slow)
    operacje = 0

    for i in range(il, 0, -1):
        lista_slow, op = sortuj(lista_slow, i, kb)
        operacje += op

    return operacje


litery = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
lista_wejsciowa = []
X0 = []
Y0 = []

for i in range(10, 1001, 100):
    lista = []
    max_dl = 0
    for j in range(i):
        slowo = ''.join(choice(litery) for _ in range(randint(4
        lista.append(slowo)
        if len(slowo) > max_dl:
            max_dl = len(slowo)
    lista_wejsciowa.append(lista)
    X0.append(len(lista))



for lista_test in lista_wejsciowa:
    ile_operacji = sortowanie_poz_slow_wizualizacja(lista_test)
    Y0.append(ile_operacji)

plt.plot(X0, Y0, label="Liczba operacji sortowania")
plt.xlabel("Liczba słów do sortowania")
plt.ylabel("Liczba przeprowadzonych operacji")
```

```
plt.legend()
plt.grid(True)
plt.show()
```

Wynikiem działania programu jest następujący wykres:



Ważne!

Złożoność czasowa tego algorytmu jest liniowa i wynosi `O(d(n + k))`, gdzie `k` to liczba różnych liter, a `d` to maksymalna długość słowa. Podstawową operacją jest tu przenoszenie elementu do kubełka lub na początek ciągu.

Już wiesz

Podsumujmy najważniejsze elementy omówione w tym e-materiale.

- Sortowanie pozycyjne jest odpowiednie do porządkowania elementów (obiektów, wielkości), w których można wyróżnić pozycje.
- Złożoność czasowa to `O(d(n + k))`.
- Sortowanie pozycyjne wymaga zastosowania dodatkowej listy pomocniczej służącej do sprawdzenia, ile liter ma najdłuższe słowo.
- Sortowanie pozycyjne słów wymaga zastosowania pomocniczego stabilnego algorytmu sortowania, np. bąbelkowego lub przez zliczanie.

Słownik

matplotlib

biblioteka służąca do przedstawienia obrazów złożonych z punktów o współrzędnych `x` oraz `y` (wykresów, histogramów, rozkładów itp.); moduł ten nie jest dostępny w standardowej instalacji języka Python – należy go zainstalować, korzystając z mechanizmu `pip`

porządek leksykograficzny

sposób porządkowania elementów zbioru analogiczny do kolejności alfabetycznej

stabilny algorytm sortowania

algorytm, który dla dwóch równych elementów w zbiorze danych wejściowych zachowuje ich kolejność przed i po sortowaniu

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Napisz program sortujący listę ciągów znaków zgodnie z porządkiem leksykograficznym, a następnie wypisujący ostatni element posortowanej listy. Skorzystaj z wybranego przez siebie stabilnego algorytmu sortowania. Zwróć uwagę, że elementy listy zapisane są wielkimi literami. Działanie swojego programu przetestuj dla następujących danych:

- `dane = ["WODA", "ZUPA", "KAWA", "LODY", "RYBA", "OWOC"]`

Specyfikacja:

Dane:

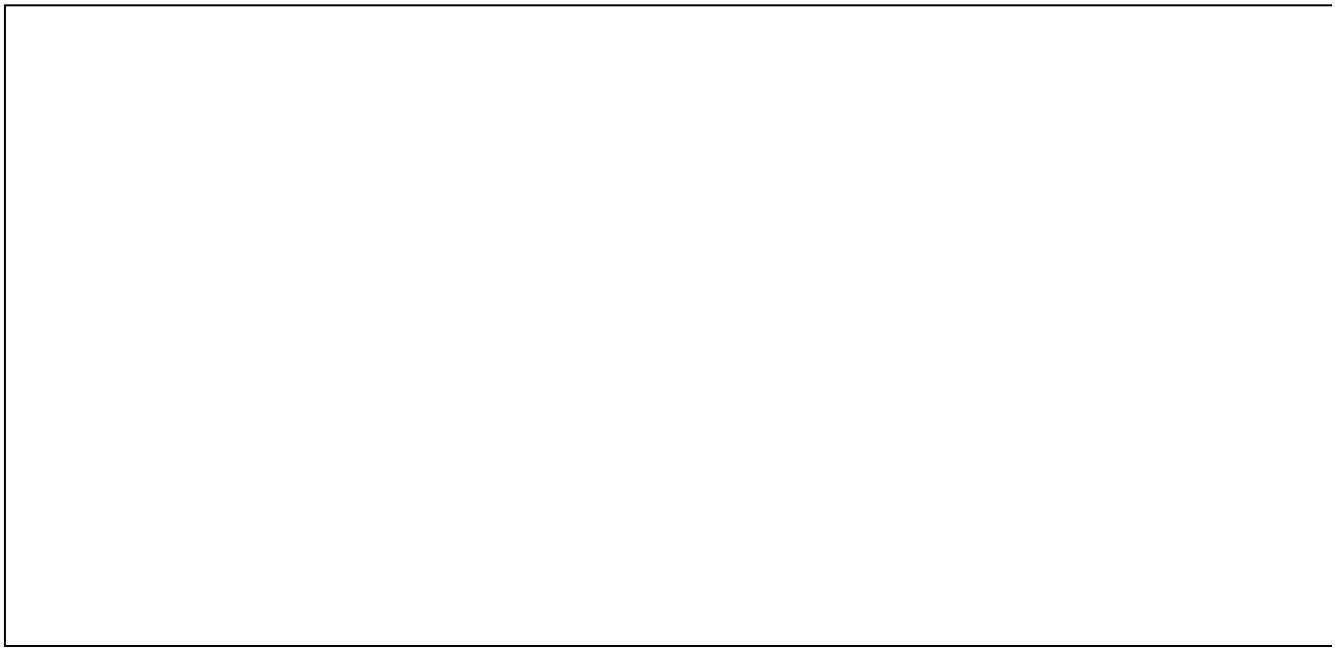
- `dane` – lista ciągów znaków do posortowania; każdy element składa się wyłącznie z dużych liter alfabetu łacińskiego

Wynik:

- `ostatni_element` – ciąg znaków; ostatni element posortowanej listy `dane`

Twoje zadania

1. Program sortuje listę `dane`, a następnie wypisuje ostatni jej element.



Ćwiczenie 2

Uzupełnij brakujące miejsca w kodzie tak, aby otrzymać działający program realizujący algorytm sortowania pozycyjnego słów. Swój program przetestuj dla listy `dane = ["magda", "ala", "adam", "ewa"]`. Elementy listy powinny zostać posortowane w porządku leksykograficznym.

Specyfikacja problemu:

Dane:

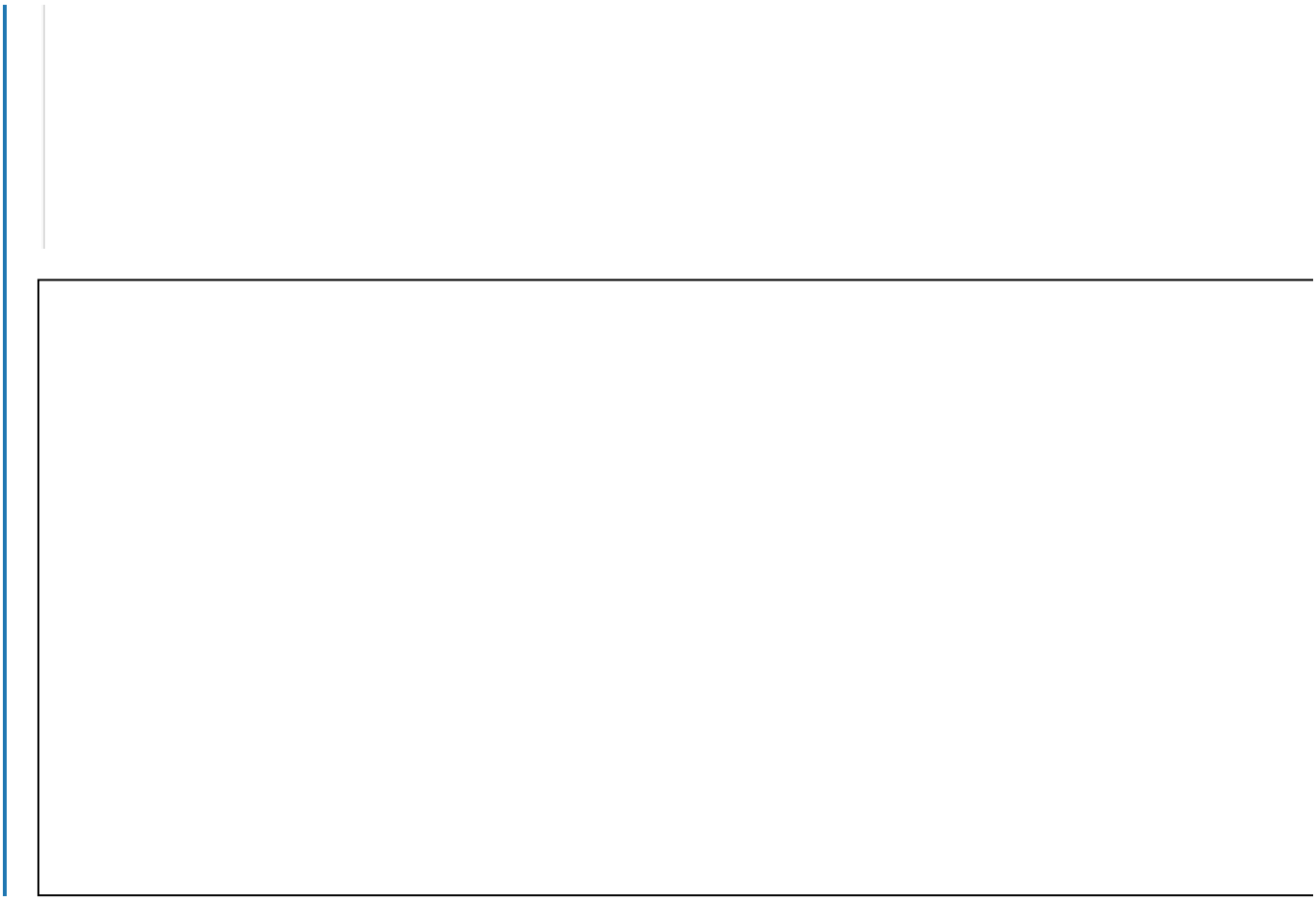
- `dane` – lista zawierająca imiona do posortowania

Wynik:

- `dane` – lista posortowana w porządku leksykograficznym

Twoje zadania

1. Zdefiniowanie funkcji `radix_sort`.
2. Funkcja zwraca listę posortowaną dla danych wejściowych `["magda", "ala", "adam", "ewa"]`



Ćwiczenie 3

Zmodyfikuj podany kod tak, aby sortował pozycyjnie ciągi znaków utworzone z cyfr. Skorzystaj w tym celu z algorytmu sortowania pozycyjnego słów, nie liczb – przykładowo: ciąg „9” jest większy od ciągu „11”. Zastosuj pomocniczo wybrany przez siebie stabilny algorytm sortowania. Lista powinna być posortowana nierosnąco.

Specyfikacja problemu:

Dane:

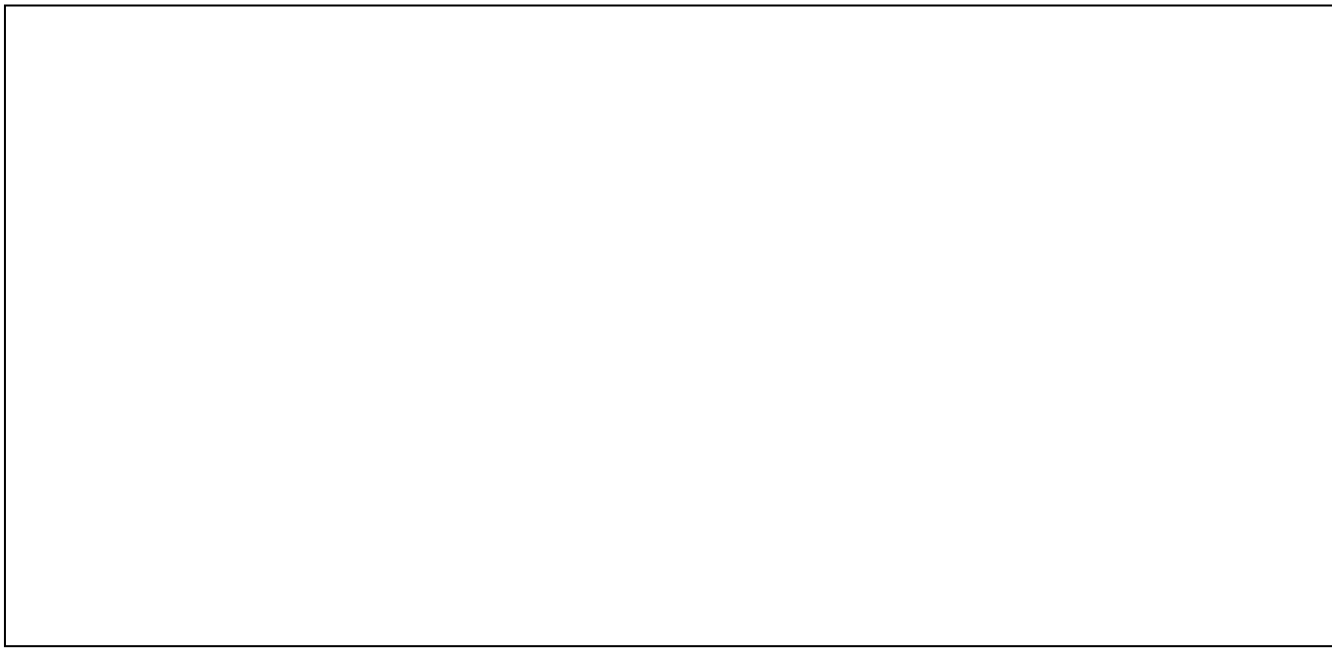
- `dane` – lista zawierająca ciągi cyfr do posortowania

Wynik:

- `dane` – lista posortowana nierosnąco

Twoje zadania

1. Zdefiniowanie funkcji `radix_sort`.
2. Funkcja zwraca listę posortowaną dla danych wejściowych `["0123", "94141", "119", "4964", "750", "0"]`



Ćwiczenie 4

Do konkursu dla startupów zgłosiło się wielu uczestników. Organizatorzy dokonali wstępnej selekcji. Przedstawiciele firm mieli prezentować swoje oprogramowanie w kolejności alfabetycznej. Jedna z przedstawicielek poprosiła o wniesienie wcześniej przygotowanej przez jej zespół makiety. Zespół techniczny poprosił o informację, która w kolejności jest prezentacja aplikacji *NoStress*. Użyj sortowania pozycyjnego słów, aby wyznaczyć indeks tej prezentacji w posortowanej liście.

Dla ułatwienia nazwy startupów zapisano małymi literami oraz bez znaków diakrytycznych. Pozbyto się również spacji.

Specyfikacja problemu:

Dane:

- `dane` – lista zawierająca nazwiska dziennikarek i dziennikarzy

Wynik:

- indeks elementu `nostress` w posortowanej liście `dane`

Twoje zadania

1. Zdefiniowanie funkcji `radix_sort`.
2. Zdefiniowanie funkcji `znajdz_nazwe`.
3. Funkcja `znajdz_nazwe` sortuje listę `dane` i zwraca liczbę: indeks nazwy "nostress" w posortowanej liście.

Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Sortowanie pozycyjne słów w języku Python

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) objaśnia dobrany algorytm, uzasadnia poprawność rozwiązania na wybranych przykładach danych i ocenia jego efektywność;

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

I + II. Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

i) struktury dynamiczne: stos, kolejka, lista (do realizacji algorytmu: ONP, symulacji problemu Flawiusza, sortowania leksykograficznego),

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Przeanalizujesz algorytm sortowania pozycyjnego słów, zaimplementowany w języku Python.
- Napiszesz w języku Python program sortujący listę imion, wykorzystując algorytm sortowania pozycyjnego.
- Zaimplementujesz algorytm wizualizujący liczbę operacji niezbędnych do wykonania sortowania pozycyjnego słów.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;

- oprogramowanie dla języka Python 3 (lub nowszej wersji), w tym PyCharm lub IDLE.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Sortowanie pozycyjne słów w języku Python". Uczniowie mają za zadanie wykonać polecenia 1 i 2 z sekcji „Film samouczek".

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć zawarte w sekcji „Wprowadzenie". Następnie wspólnie z uczniami ustala kryteria sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel zadaje uczniom pytanie dotyczące ich aktualnego stanu wiedzy w obszarze poruszanego tematu, opartego o programowanie.

Faza realizacyjna:

1. **Praca z multimedium.** Nauczyciel sprawdza przygotowanie uczniów do lekcji, prosząc wybrane osoby o przedstawienie rozwiązania polecenia 1 z sekcji „Film samouczek". Jeżeli przygotowanie uczniów jest niewystarczające, ponownie zapoznają się oni z filmem z tej sekcji.
2. **Praca z tekstem.** Nauczyciel wyświetla zawartość sekcji „Przeczytaj". Na forum klasy uczniowie analizują przedstawione w niej treści, a w kolejnym kroku testują omówione programy na swoich komputerach.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenie nr 1 z sekcji „Sprawdź się", a następnie porównują swoje odpowiedzi z kolegą lub koleżanką.
4. W kolejnym etapie uczniowie dobierają się w pary i wykonują ćwiczenia nr 2–3 z sekcji „Sprawdź się". Następnie konsultują swoje rozwiązania z inną parą uczniów i ustalają jedną wersję odpowiedzi.

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat i cele lekcji zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji następuje omówienie ewentualnych problemów z rozwiązaniem ćwiczeń z sekcji „Sprawdź się".
2. Wybrany uczeń podsumowuje zajęcia z programowania w Pythonie, zwracając uwagę na nabyte umiejętności.

Praca domowa:

1. Uczniowie wykonują ćwiczenie 4 z sekcji „Sprawdź się".
2. Uczniowie wykonują polecenie 3 i 4 z sekcji „Film samouczek".

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka Python 3 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania PyCharm lub IDLE.

Wskazówki metodyczne:

- Uczniowie mogą wykorzystać treści w sekcjach: „Film samouczek”, „Przeczytaj”, „Sprawdź się” jako materiał do lekcji powtórkowej.