Sortowanie pozycyjne liczb w języku Python

Źródło: Jan Antonin Kolar, domena publiczna.

Algorytm sortowania pozycyjnego (ang. *radix sort*) jest jednym z podstawowych sposobów porządkowania danych. Szczególnie dobrze sprawdza się przy sortowaniu dużego zbioru liczb należących do niewielkiego zakresu. Najważniejsze informacje na ten temat przedstawiliśmy w e-materiale Sortowanie pozycyjne liczb. Teraz zajmiemy się implementacją tego algorytmu w języku Python.

Implementację sortowania szybkiego w innych językach programowania omawiamy w e-materiałach:

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

Więcej zadań? Przejdź do: Sortowanie pozycyjne liczb – zadania maturalne.

Twoje cele

- Przeanalizujesz implementację algorytmu sortowania pozycyjnego liczb w języku Python.
- Zaimplementujesz w języku Python różne warianty algorytmu sortowania pozycyjnego liczb, wykorzystujące pomocniczo sortowanie kubełkowe, sortowanie bąbelkowe, a także sortowanie przez zliczanie.

- Wykonasz kilka ćwiczeń z programowania dotyczących algorytmu sortowania pozycyjnego.

Przeczytaj

Przykład 1

Przygotujmy program, który zrealizuje sortowanie pozycyjne liczb. Założymy, że liczby w danych wejściowych nie muszą być tej samej długości, więc jest konieczne przygotowanie funkcji pomocniczej, która sprawdzi, z ilu cyfr składa się największa liczba – będziemy to robić, wykonując dzielenie całkowite liczby przez kolejne potęgi liczby 10, dopóki iloraz nie będzie równy 0. Następnie, zaczynając od cyfry jedności, będziemy stopniowo sortować liczby po kolejnych pozycjach.

Specyfikacja problemu:

Dane:

- `lista_liczb` – lista zawierająca liczby całkowite; lista do posortowania

Wynik:

- `posortowana_lista` – lista zawierająca liczby całkowite; posortowana lista wejściowa

Zaczniemy od znalezienia maksymalnego podzielnika. Jest to największa potęga liczby 10, mniejsza od maksymalnej wartości z listy. Użyjemy do tego funkcji:

```
def sprawdz_maksimum(lista_liczb):
    maximum = lista_liczb[0]
    for i in range(1, len(lista_liczb)):
        if lista_liczb[i] > maximum:
            maximum = lista_liczb[i]
    podzielnik = 1

    while maximum > 0:
        maximum = maximum // 10
        if maximum > 0:
            podzielnik *= 10
    return podzielnik
```

Kolejnym krokiem będzie napisanie funkcji pomocniczej sortującej liczby na danej pozycji. Potrzebujemy dodatkowej metody sortowania, która musi być stabilna, aby liczby zachowały kolejność z sortowań na mniej znaczących pozycjach. W tym przypadku korzystamy z sortowania kubełkowego, ponieważ jego złożoność jest liniowa. Liczby będziemy umieszczać w kubełkach o wartości równej ich cyfrze na aktualnie sortowanej

pozycji, a następnie połączymy kubełki w jedną listę. Jako parametr funkcji będziemy przekazywać podzielnik, który będzie odpowiadał pozycji, po której chcemy sortować (np. dla cyfry jedności: 1, dla cyfry dziesiątek: 10 itd.).

```
def sortuj_liczby(lista, podzielnik):
    kubelki = [None] * 10
    posortowane_po_pozycji = []
    for element in lista:
        # dodajemy element do listy znajdującej się na indeksie
        # jego wartości na aktualnie sprawdzanej pozycji
        wart = (element // podzielnik) % 10
        if type(kubelki[wart]) is list:
            kubelki[wart].append(element)
        else:
            kubelki[wart] = [element]
    for element in kubelki:
        # przepisujemy do nowej listy wszystkie elementy po kol
        # z racji powyższego podziału będą już posortowane po d
        while type(element) is list and len(element) > 0:
            posortowane_po_pozycji.append(element.pop(0))
    return posortowane_po_pozycji
```

Aby cała lista została posortowana, musimy posortować liczby po każdej pozycji. Zrobimy to w ten sposób:

```
podzielnik = sprawdz_maksimum(lista_liczb)

# sortujemy dla każdego możliwego podzielnika
p = 1
while p <= podzielnik:
    lista_liczb = sortuj_liczby(lista_liczb, p)
    p *= 10
```

Następnie połączymy cały program w jedną funkcję i przetestujemy go dla przykładowych danych:

```
def posortuj_pozycyjnie_liczby(lista_liczb):
    def sprawdz_maksimum(lista_liczb):
        maximum = lista_liczb[0]
        for i in range(1, len(lista_liczb)):
            if lista_liczb[i] > maximum:
                maximum = lista_liczb[i]
```

```
        podzielnik = 1

        while maximum > 0:
            maximum = maximum // 10
            if maximum > 0:
                podzielnik *= 10
        return podzielnik

    def sortuj_liczby(lista, podzielnik):
        kubelki = [None] * 10
        posortowane_po_pozycji = []
        for element in lista:
            # dodajemy element do listy znajdującej się na inde
            # jego wartości na aktualnie sprawdzanej pozycji
            wart = (element // podzielnik) % 10
            if type(kubelki[wart]) is list:
                kubelki[wart].append(element)
            else:
                kubelki[wart] = [element]
        for element in kubelki:
            # przepisujemy do nowej listy wszystkie elementy po
            # z racji powyższego podziału będą już posortowane
            while type(element) is list and len(element) > 0:
                posortowane_po_pozycji.append(element.pop(0))
        return posortowane_po_pozycji

    podzielnik = sprawdz_maksimum(lista_liczb)

    # sortujemy dla każdego możliwego podzielnika
    p = 1
    while p <= podzielnik:
        lista_liczb = sortuj_liczby(lista_liczb, p)
        p *= 10

    return lista_liczb

# przykład wywołania
l = [321, 3476, 577, 85, 294, 235, 4565, 34, 547654]
posortowana_lista = posortuj_pozycyjnie_liczby(l)
print(posortowana_lista)
#[34, 85, 235, 294, 321, 577, 3476, 4565, 547654]

```

```
q = [1693, 1024, 3080, 2112, 1128, 3772, 907, 1719, 3816, 2530,
posortowana_lista = posortuj_pozycyjnie_liczby(q)
print(posortowana_lista)
#[15, 104, 179, 256, 292, 470, 594, 907, 918, 1014, 1024, 1061,
```

Ważne!

Wyjaśnijmy zapis `element.pop(0)`. Metoda `pop()` usuwa i zwraca ostatni element listy, ale my – w celu zachowania kolejności elementów odkładanych do kubełka – pobieramy pierwszy element z listy, jednocześnie go z niej usuwając, zatem używamy metody `pop` z parametrem (jeśli chcesz dowiedzieć się więcej, informacje na ten temat znajdziesz w dostępnej w internecie dokumentacji języka Python).

Przykład 2

Korzystając z funkcji zaimplementowanej w przykładzie 1, stworzymy program pokazujący wyniki sortowania losowej listy liczb dla poszczególnych pozycji oraz wizualizujący (za pomocą modułu matplotlib) zależność liczby operacji od długości wejściowej tablicy danych oraz od wartości największego elementu w liście.

Specyfikacja problemu:

Dane:

- `lista_liczb` – lista zawierająca liczby całkowite; lista do posortowania

Wynik:

- wykres przedstawiający liczbę operacji ze względu na długość listy startowej, a także maksymalną wartość w danej liście

Program przetestujemy dla list o długościach wielokrotności liczby 10 z zakresu $\langle 10,\ 100 \rangle$ o losowych wartościach z zakresu $\langle 1,\ 4000 \rangle$.

W tym celu w rozwiązaniu z poprzedniego zadania dodamy do funkcji sortującej zmienną zliczającą wykonane operacje, którą będziemy inkrementować za każdym razem, gdy będziemy dodawać lub wyciągać elementy z listy. Poza tym napiszemy kod, który przy użyciu funkcji `scatter()` i `plot()` z biblioteki `matplotlib` wyświetli interesujące nas dane.

```
from random import randint
import matplotlib.pyplot as plt

X0 = [x for x in range(10, 101, 10)]
```

```
Y0 = []
Y1 = []

for x in X0:
    l_test = [randint(1, 4000) for y in range(x)]
    l, p = posortuj_pozycyjnie_liczby_wizualizacja(l_test)
    Y0.append(p)
    Y1.append(max(l))
    lista, oper = posortuj_pozycyjnie_liczby_wizualizacja(l_tes

plt.plot(X0, Y0, label="Liczba operacji sortowania")
plt.scatter(X0, Y1, label="Maksymalna wartość w liście")
plt.xlabel("Liczba liczb do sortowania")
plt.ylabel("Liczba przeprowadzonych operacji")
plt.legend()
plt.grid(True)
plt.show()
```

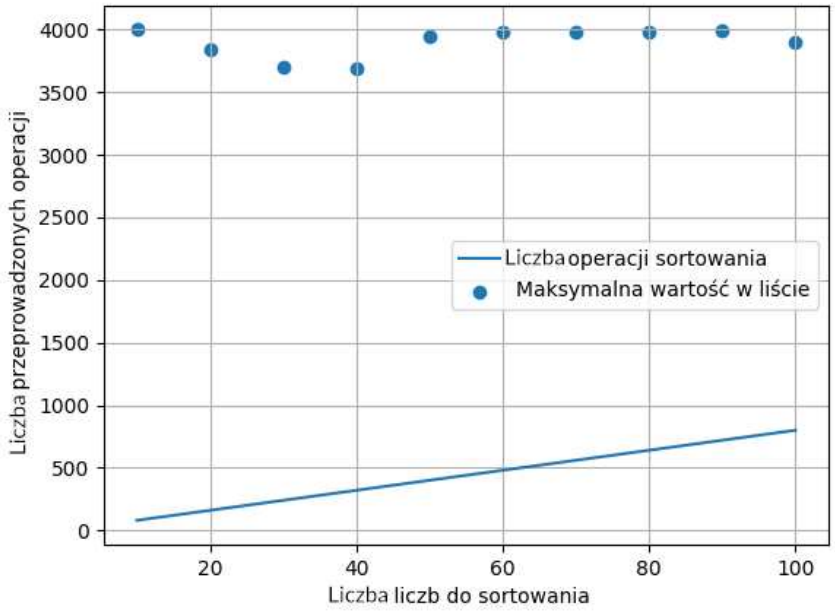
Cały program prezentuje się następująco:

```
def posortuj_pozycyjnie_liczby_wizualizacja(lista_liczb):
    def sprawdz_maksimum(lista_liczb):
        maximum = lista_liczb[0]
        for i in range(1, len(lista_liczb)):
            if lista_liczb[i] > maximum:
                maximum = lista_liczb[i]
        podzielnik = 1

        while maximum > 0:
            maximum = maximum // 10
            if maximum > 0:
                podzielnik *= 10
        return podzielnik

    def sortuj_liczby(lista, podzielnik):
        operacje = 0
        kubelki = [""] * 10
        posortowane_po_pozycji = []
        for element in lista:
            wart = (element // podzielnik) % 10
            operacje += 1
            if type(kubelki[wart]) is list:
```

```
                kubelki[wart].append(element)
            else:
                kubelki[wart] = [element]
        for element in kubelki:
            while type(element) is list and len(element) > 0:
                posortowane_po_pozycji.append(element.pop(0))
                operacje += 1
        return posortowane_po_pozycji, operacje

    podzielnik = sprawdz_maksimum(lista_liczb)

    # sortujemy dla każdego możliwego podzielnika
    p = 1
    suma_operacji = 0
    while p <= podzielnik:
        lista_liczb, operacje = sortuj_liczby(lista_liczb, p)
        suma_operacji += operacje
        p *= 10

    return lista_liczb, suma_operacji


from random import randint
import matplotlib.pyplot as plt

X0 = [x for x in range(10, 101, 10)]
Y0 = []
Y1 = []

for x in X0:
    l_test = [randint(1, 4000) for y in range(x)]
    l, p = posortuj_pozycyjnie_liczby_wizualizacja(l_test)
    Y0.append(p)
    Y1.append(max(l))
    lista, oper = posortuj_pozycyjnie_liczby_wizualizacja(l_tes

plt.plot(X0, Y0, label="Liczba operacji sortowania")
plt.scatter(X0, Y1, label="Maksymalna wartość w liście")
plt.xlabel("Liczba liczb do sortowania")
plt.ylabel("Liczba przeprowadzonych operacji")
plt.legend()
plt.grid(True)
```

```
65 plt.show()
```

Program powinien narysować wykres wyglądający podobnie do tego:



Na wykresie można zauważyć, że każda lista zawierała maksymalną liczbę większą niż 1000, a co za tym idzie, za każdym razem trzeba było wykonać sortowanie na czterech pozycjach. Przy okazji potwierdziliśmy doświadczalnie, że złożoność algorytmu jest liniowa względem liczby sortowanych liczb, co widać po liczbie operacji potrzebnych do posortowania listy.

Już wiesz

Podsumujmy najważniejsze wiadomości:

- sortowanie pozycyjne jest odpowiednie do porządkowania elementów (obiektów, wielkości), w których można wyróżnić pozycje;
- sortowanie pozycyjne wymaga zastosowania dodatkowego stabilnego algorytmu sortującego;
- złożoność czasowa to $O\left(d\left(n+k\right)\right)$;
- sortowanie pozycyjne jest stabilnym algorytmem sortowania.

Słownik

matplotlib

biblioteka służąca do przedstawiania obrazów składających się z punktów o współrzędnych `x` oraz `y` (wykresów, histogramów, rozkładów itp.); moduł `matplotlib`

nie jest dostępny w standardowej instalacji języka Python – należy go zainstalować, korzystając z mechanizmu `pip`

stabilny algorytm sortowania

typ algorytmów sortujących, które w przypadku występowania kilku obiektów o takiej samej wartości w zbiorze wejściowym, po posortowaniu zachowują ich wzajemną kolejność początkową

Prezentacja multimedialna

Polecenie 1

W sekcji „Przeczytaj” analizowaliśmy program sortujący pozycyjnie liczby z zastosowaniem pomocniczego algorytmu sortowania kubełkowego. Zapoznaj się z prezentacją przedstawiającą jego modyfikację, dzięki której wyświetlone zostaną kolejne etapy sortowania oraz te kubełki, które zawierają więcej niż jeden element.

W tym e-materiale wykorzystujemy pomocnicze sortowania – kubełkowe oraz bąbelkowe. Więcej informacji na ich temat znajdziesz w e-materiałach:

- Sortowanie bąbelkowe,
- Sortowanie kubełkowe.

1

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PuZcry19d

Do sortowania użyjemy funkcji opracowanej w sekcji „Przeczytaj”. Zmodyfikujemy ją w taki sposób, aby wyświetlać kolejne etapy sortowania oraz te kubełki, które zawierają więcej niż jeden element.

```
1 def posortuj_pozycyjnie_liczby_kubelek(lista_liczb):
2
3     def sprawdz_maksimum(lista_liczb):
4         maksimum = lista_liczb[0]
5         for i in range(1, len(lista_liczb)):
6             if lista_liczb[i] > maksimum:
```

```
                maksimum =
lista_liczb[i]
        podzielnik = 1

        while maksimum >
0:
            maksimum =
maksimum // 10
            if maksimum >
0:
                podzielnik
*= 10
        return podzielnik


    def
sortuj_liczby(lista,
podzielnik):
        op = 0

print("Przetwarzamy dla
podzielnika", podzielnik,
".")
        # tworzymy kubełki
        lista_wyjsciowa =
[[] for _ in range(10)]
        for element in
lista:
            wart =
(element // podzielnik) %
10
            op += 1

lista_wyjsciowa[wart].appe
nd(element)
        for element in
lista_wyjsciowa:
            if
len(element) > 0:

print("Kubełek zawiera :
", element)

        # wypakowanie
kubełków
```

```
        lista_wyjsciowa =
[element for kubelek in
lista_wyjsciowa for
element in kubelek]
        op +=
len(lista_wyjsciowa)
        print("Lista ",
lista_wyjsciowa," wymagała
", op," operacji.")
        return
lista_wyjsciowa

    podzielnik =
sprawdz_maksimum(lista_lic
zb)

    # sortujemy dla
każdego możliwego
podzielnika
    p = 1
    while p <= podzielnik:
        lista_liczb =
sortuj_liczby(lista_liczb,
p)
        p *= 10

    return lista_liczb
```

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PuZcry19d

Ustalamy dane wejściowe, a więc listę liczb:

```
lista_wejsciowa = [667,
6906, 8247, 1699, 5984,
8050, 1783, 6707, 9399,
664, 6987, 3360, 914,
9695, 1435, 723, 775,
1415, 8227, 6421, 2590,
2754, 2466, 8541, 8407]
```

3

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PuZcry19d

Uruchamiamy zdefiniowaną funkcję sortującą:

```
lista_wejsciowa = [667,
6906, 8247, 1699, 5984,
8050, 1783, 6707, 9399,
664, 6987, 3360, 914,
9695, 1435, 723, 775,
1415, 8227, 6421, 2590,
2754, 2466, 8541, 8407]
print(posortuj_pozycyjnie_
liczby_kubelek(lista_wejsc
iowa))
```

4

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PuZcry19d

Wewnątrz funkcji z pierwszego slajdu w linii o numerze 36 wykona się funkcja sprawdzająca maksymalny podzielnik, który zależy od tego, ile cyfr zawiera największa z liczb w liście:

```
podzielnik =
sprawdz_maksimum(lista_lic
zb)
```

5

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PuZcry19d

Podzielnik wynosi 1000, gdyż największa liczba jest czterocyfrowa, a więc podzielona całkowitoliczbowo przez 1000 daje liczbę większą od zera.

6

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PuZcry19d

W funkcji z pierwszego slajdu użyliśmy następującego kodu. Dla każdego podzielnika wykonujemy sortowanie. Stosujemy pętlę `while`, za każdym przejściem zwiększając podzielnik dziesięciokrotnie:

```
1 p = 1
2 while p <= podzielnik:
3     lista_liczb =
  sortuj_liczby(lista_liczb,
  p)
4     p *= 10
```

7

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PuZcry19d

W pierwszym sortowaniu wykona się funkcja `sortuj_liczby()`, która w kubełkach będzie przechowywała liczby z tą samą cyfrą na pozycji jedności:

```
1 Przetwarzamy dla
  podzielnika 1.
2 Kubełek zawiera : [8050,
  3360, 2590]
3 Kubełek zawiera : [6421,
  8541]
```

```
Kubełek zawiera : [1783, 723]
Kubełek zawiera : [5984, 664, 914, 2754]
Kubełek zawiera : [9695, 1435, 775, 1415]
Kubełek zawiera : [6906, 2466]
Kubełek zawiera : [667, 8247, 6707, 6987, 8227, 8407]
Kubełek zawiera : [1699, 9399]
```

8

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PuZcry19d

Po złączeniu kolejno elementów tego przebiegu uzyskujemy listę:

```
[8050, 3360, 2590, 6421, 8541, 1783, 723, 5984, 664, 914, 2754, 9695, 1435, 775, 1415, 6906, 2466, 667, 8247, 6707, 6987, 8227, 8407, 1699, 9399]
```

Wykonanie tego sortowania zajęło 50 operacji: 25 operacji rozkładania liczb do kubełków oraz 25 operacji wyciągania liczb z kubełków i scalenia ich w jedną listę.

9

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PuZcry19d

W drugim sortowaniu wykona się funkcja `sortuj_liczby()`, która w kubełkach będzie przechowywała liczby z tą samą cyfrą na pozycji dziesiątek:

```
Przetwarzamy dla podzielnika 10.
Kubełek zawiera : [6906, 6707, 8407]
Kubełek zawiera : [914, 1415]
Kubełek zawiera : [6421, 723, 8227]
Kubełek zawiera : [1435]
Kubełek zawiera : [8541, 8247]
Kubełek zawiera : [8050, 2754]
Kubełek zawiera : [3360, 664, 2466, 667]
Kubełek zawiera : [775]
Kubełek zawiera : [1783, 5984, 6987]
Kubełek zawiera : [2590, 9695, 1699, 9399]
```

10

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PuZcry19d

Po złączeniu kolejno elementów tego przebiegu uzyskujemy listę:

```
[6906, 6707, 8407, 914, 1415, 6421, 723, 8227, 1435, 8541, 8247, 8050, 2754, 3360, 664, 2466, 667, 775, 1783, 5984, 6987, 2590, 9695, 1699, 9399]
```

Wykonanie tego sortowania zajęło 50 operacji: 25 operacji rozkładania liczb do kubełków oraz 25 operacji wyciągania liczb z kubełków i scalenia ich w jedną listę.

11

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PuZcry19d

W trzecim sortowaniu wykona się funkcja `sortuj_liczby()`, która w kubełkach będzie przechowywała liczby z tą samą cyfrą na pozycji setek:

```
Przetwarzamy dla podzielnika 100.
Kubełek zawiera : [8050]
Kubełek zawiera : [8227, 8247]
Kubełek zawiera : [3360, 9399]
Kubełek zawiera : [8407, 1415, 6421, 1435, 2466]
Kubełek zawiera : [8541, 2590]
Kubełek zawiera : [664, 667, 9695, 1699]
Kubełek zawiera : [6707, 723, 2754, 775, 1783]
Kubełek zawiera : [6906, 914, 5984, 6987]
```

12

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PuZcry19d

Po złączeniu kolejno elementów tego przebiegu uzyskujemy listę:

```
[8050, 8227, 8247, 3360,
9399, 8407, 1415, 6421,
1435, 2466, 8541, 2590,
664, 667, 9695, 1699,
6707, 723, 2754, 775,
1783, 6906, 914, 5984,
6987]
```

Wykonanie tego sortowania zajęło 50 operacji: 25 operacji rozkładania liczb do kubełków oraz 25 operacji wyciągania liczb z kubełków i scalenia ich w jedną listę.

13

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PuZcry19d

W czwartym sortowaniu wykona się funkcja `sortuj_liczby()`, która w kubełkach będzie przechowywała liczby z tą samą cyfrą na pozycji tysięcy:

```
Przetwarzamy dla
podzielnika 1000.
Kubełek zawiera : [664,
667, 723, 775, 914]
Kubełek zawiera : [1415,
1435, 1699, 1783]
Kubełek zawiera : [2466,
2590, 2754]
Kubełek zawiera : [3360]
Kubełek zawiera : [5984]
Kubełek zawiera : [6421,
6707, 6906, 6987]
Kubełek zawiera : [8050,
8227, 8247, 8407, 8541]
Kubełek zawiera : [9399,
9695]
```

14

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PuZcry19d

Po złączeniu kolejno elementów tego przebiegu uzyskujemy listę:

```
[664, 667, 723, 775, 914,
1415, 1435, 1699, 1783,
2466, 2590, 2754, 3360,
5984, 6421, 6707, 6906,
6987, 8050, 8227, 8247,
8407, 8541, 9399, 9695]
```

Wykonanie tego sortowania zajęło 50 operacji: 25 operacji rozkładania liczb do kubełków oraz 25 operacji wyciągania liczb z kubełków i scalenia ich w jedną listę.

15

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PuZcry19d

Lista jest posortowana:

```
[664, 667, 723, 775, 914,
1415, 1435, 1699, 1783,
2466, 2590, 2754, 3360,
5984, 6421, 6707, 6906,
6987, 8050, 8227, 8247,
8407, 8541, 9399, 9695]
```

Wykonanie tego sortowania zajęło cztery razy po 50 operacji – liczba operacji wzrasta o stałą liczbę wraz z liczbą potrzebnych sortowań (czyli maksymalną długością liczby w liście). Widzimy więc, że złożoność tego algorytmu jest liniowa i zależy od liczby elementów w liście oraz liczby cyfr najdłuższego elementu.

Cały kod programu z prezentacji wygląda następująco:

```
def posortuj_pozycyjnie_liczby_kubelek(lista_liczb):

    def sprawdz_maksimum(lista_liczb):
        maksimum = lista_liczb[0]
        for i in range(1, len(lista_liczb)):
            if lista_liczb[i] > maksimum:
                maksimum = lista_liczb[i]
        podzielnik = 1

        while maksimum > 0:
            maksimum = maksimum // 10
            if maksimum > 0:
                podzielnik *= 10
        return podzielnik


    def sortuj_liczby(lista, podzielnik):
        op = 0
        print("Przetwarzamy dla podzielnika", podzielnik, ".")
        # tworzymy kubełki
        lista_wyjsciowa = [[] for _ in range(10)]
        for element in lista:
            wart = (element // podzielnik) % 10
            op += 1
            lista_wyjsciowa[wart].append(element)
        for element in lista_wyjsciowa:
            if len(element) > 0:
                print("Kubełek zawiera : ", element)

        # wypakowanie kubełków
        lista_wyjsciowa = [element for kubelek in lista_wyjsciowa
        op += len(lista_wyjsciowa)
        print("Lista ", lista_wyjsciowa," wymagała ", op," operac
        return lista_wyjsciowa

    podzielnik = sprawdz_maksimum(lista_liczb)

```

```
    # sortujemy dla każdego możliwego podzielnika
    p = 1
    while p <= podzielnik:
        lista_liczb = sortuj_liczby(lista_liczb, p)
        p *= 10

    return lista_liczb

lista_wejsciowa = [667, 6906, 8247, 1699, 5984, 8050, 1783, 6707,
print(posortuj_pozycyjnie_liczby_kubelek(lista_wejsciowa))
```

Polecenie 2

Zapisz program sortujący pozycyjnie liczby, który wykorzystuje pomocniczo sortowanie bąbelkowe.

Działanie programu przetestuj dla następujących danych:

```
dane = [5000, 23, 567, 34909, 2, 98, 1010, 8, 90]
```

Specyfikacja problemu:

Dane:

- `dane` – nieposortowana tablica liczb naturalnych

Wynik:

- `dane` – posortowana niemalejąco tablica liczb naturalnych

```

```

```

```

Polecenie 3

Porównaj swoje rozwiązanie z prezentacją przedstawiającą implementację algorytmu sortowania pozycyjnego liczb z wykorzystaną pomocniczo metodą sortowania bąbelkowego.

1

Najpierw zaimplementujemy funkcję, która będzie sortować bąbelkowo po zadanej pozycji liczby:

```
def sortowanie_babelkowe(dane, podzielnik):
    n = len(dane)
    swapped = False
    for i in range(n - 1):
        for j in range(0, n - i - 1):
            if (dane[j] // podzielnik) % 10 > (dane[j + 1] // podzielnik) % 10:
                swapped = True
```

```
                dane[j],
dane[j + 1] = dane[j + 1],
dane[j]
        if not swapped:
            return
```

2

Ustalimy dane wejściowe i przetestujemy sortowanie bąbelkowe dla cyfry dziesiątek:

```
dane = [5000, 23, 567,
34909, 2, 98, 1010, 8, 90]
sortowanie_babelkowe(dane,
10)
print(dane)
# [5000, 34909, 2, 8,
1010, 23, 567, 98, 90]
```

3

Teraz przejdziemy do implementacji głównej funkcji odpowiadającej za sortowanie pozycyjne. W tym celu skorzystamy z wcześniej zaimplementowanej funkcji do szukania maksymalnego podzielnika:

```
def
sprawdz_maksimum(lista_lic
zb):
        maksimum =
lista_liczb[0]
        for i in range(1,
len(lista_liczb)):
            if
lista_liczb[i] > maksimum:
                maksimum =
lista_liczb[i]
        podzielnik = 1

```

```
        while maksimum > 0:
            maksimum = maksimum // podzielnik
            if maksimum > 0:
                podzielnik *= 10
        return podzielnik
```

4

Główna funkcja programu będzie sortować bąbelkowo dla każdej możliwej pozycji. Implementacja jest również analogiczna do poprzedniego przykładu.

```
def sortowanie_pozycyjne(dane):
    podzielnik = sprawdz_maksimum(dane)
    p = 1
    while p <= podzielnik:
        sortowanie_babelkowe(dane, podzielnik)
        p *= 10
    return dane

```

5

Wszystkie potrzebne funkcje wyglądają następująco:

```
def sortowanie_babelkowe(dane, podzielnik):
    n = len(dane)
```

```
    swapped = False
    for i in range(n - 1):
        for j in range(0, n - i - 1):
            if (dane[j] // podzielnik) % 10 > (dane[j + 1] // podzielnik) % 10:
                swapped = True
                dane[j], dane[j + 1] = dane[j + 1], dane[j]
        if not swapped:
            return


def sprawdz_maksimum(lista_liczb):
        maksimum = lista_liczb[0]
        for i in range(1, len(lista_liczb)):
            if lista_liczb[i] > maksimum:
                maksimum = lista_liczb[i]
        podzielnik = 1

        while maksimum > 0:
            maksimum = maksimum // podzielnik
            if maksimum > 0:
                podzielnik *= 10
        return podzielnik


def sortowanie_pozycyjne(dane):
    podzielnik = sprawdz_maksimum(dane)
```

```
    p = 1
    while p <= podzielnik:

sortowanie_babelkowe(dane, podzielnik)
        p *= 10
    return dane
```

6

Przetestujemy działanie dla wcześniej zdefiniowanej listy `dane`:

```
def sortowanie_babelkowe(dane, podzielnik):
    n = len(dane)
    swapped = False
    for i in range(n - 1):
        for j in range(0, n - i - 1):
            if (dane[j] // podzielnik) % 10 > (dane[j + 1] // podzielnik) % 10:
                swapped = True
                dane[j], dane[j + 1] = dane[j + 1], dane[j]
        if not swapped:
            return


def sprawdz_maksimum(lista_liczb):
        maksimum = lista_liczb[0]
        for i in range(1, len(lista_liczb)):
            if lista_liczb[i] > maksimum:
```

```
17                  maksimum =
lista_liczb[i]
18          podzielnik = 1
19
20          while maksimum >
0:
21              maksimum =
maksimum // podzielnik
22              if maksimum >
0:
23                  podzielnik
*= 10
24          return podzielnik
25
26
27  def
sortowanie_pozycyjne(dane)
:
28      podzielnik =
sprawdz_maksimum(dane)
29      p = 1
30      while p <= podzielnik:
31
sortowanie_babelkowe(dane,
podzielnik)
32          p *= 10
33      return dane
34
35  dane = [5000, 23, 567,
34909, 2, 98, 1010, 8, 90]
36  print(sortowanie_pozycyjne
(dane))
37  # [2, 8, 23, 90, 98, 567,
1010, 5000, 34909]
```


W algorytmie sortowania pozycyjnego możemy skorzystać z dowolnego sortowania, które jest stabilne. Niestety sortowanie bąbelkowe nie jest optymalnym wyborem, ponieważ jego złożoność to $O\left(n^2\right)$, a musimy wykonać je m razy, gdzie m

to długość liczby z listy o największej liczbie znaków. Lepszym wyborem będzie zastosowanie sortowania przez zliczanie, którego złożoność to $O(n+k)$. W tym przypadku k będzie równe 10, więc złożoność całego sortowania pozycyjnego to $O(n*m)$.

Polecenie 4

Zapisz program sortujący pozycyjnie liczby, który wykorzystuje pomocniczo algorytm sortowania przez zliczanie.

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Wskaż zdanie prawdziwe

- W algorytmie sortowania pozycyjnego liczb porządkujemy elementy składające się z wielu pozycji, zaczynając od tych najmniej znaczących, przechodząc stopniowo do najbardziej znaczących. Dla liczb naturalnych najmniej znacząca pozycja to pozycja jedności – pierwsza w ciągu cyfr.
- W algorytmie sortowania pozycyjnego liczb porządkujemy elementy składające się z wielu pozycji, zaczynając od tych najmniej znaczących, przechodząc stopniowo do najbardziej znaczących. Dla liczb naturalnych najmniej znacząca pozycja to pozycja jedności – ostatnia w ciągu cyfr.
- W algorytmie sortowania pozycyjnego liczb porządkujemy elementy składające się z wielu pozycji, zaczynając od tych najbardziej znaczących, przechodząc stopniowo do najmniej znaczących. Dla liczb naturalnych najbardziej znacząca pozycja to pozycja jedności – ostatnia w ciągu cyfr.

Ćwiczenie 2

Uzupełnij luki w odpowiednich miejscach w kodzie, tak aby cyfry na kolejnych pozycjach były sortowane metodą sortowania przez zliczanie. Zbiór ma być posortowany niemalejąco.

Specyfikacja problemu:

Dane:

- `dane` – lista zawierająca liczby całkowite; lista do posortowania

Wynik:

- `posortowane` – lista zawierająca liczby całkowite; posortowana lista wejściowa

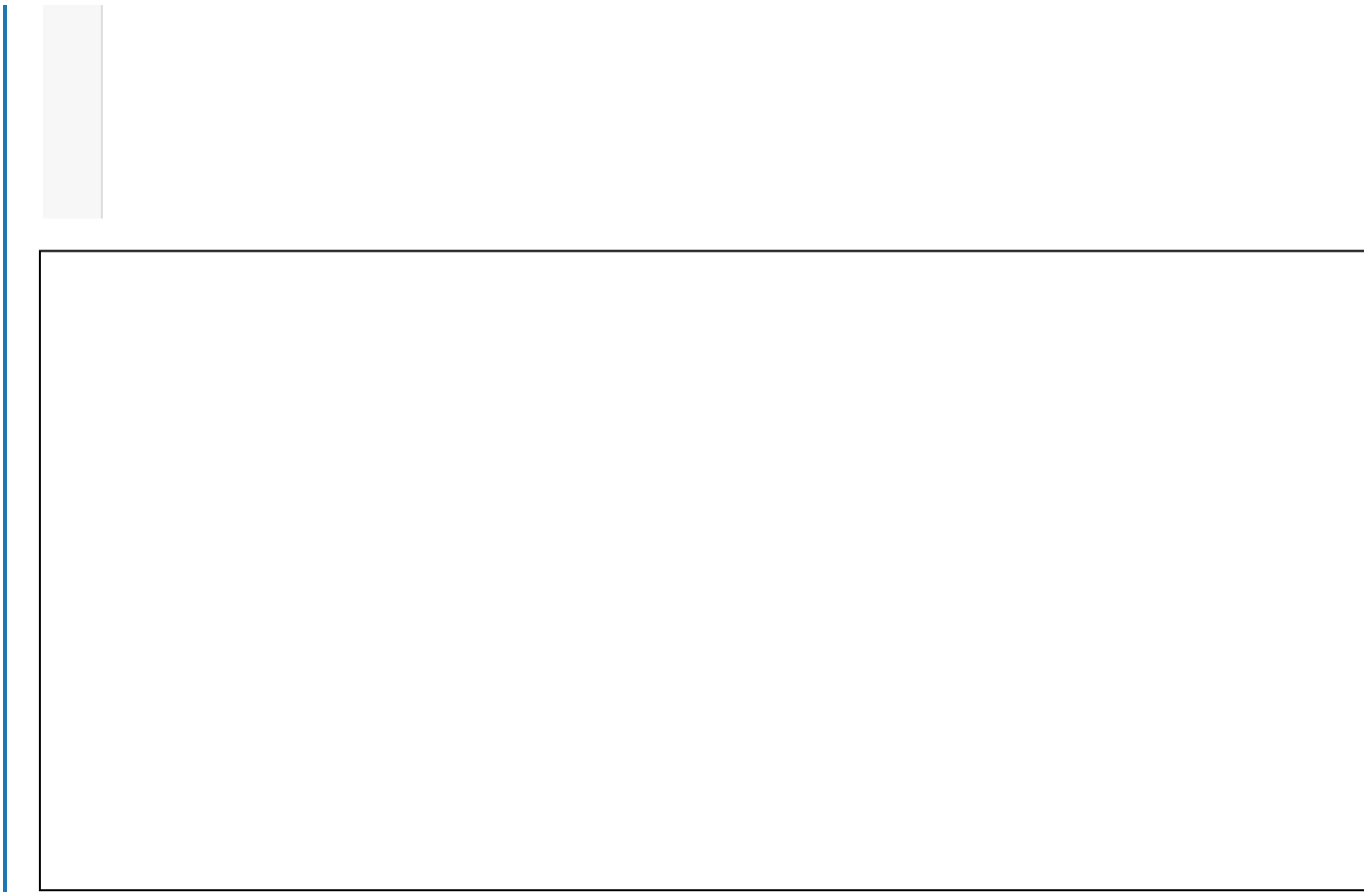
Program przetestuj dla listy: `[876,111,412,908,765,321,765,23,3]`

Twoje zadania

1. Program sortuje listę liczb niemalejąco, wykorzystując połączenie algorytmów sortowania pozycyjnego i sortowania przez zliczanie.

```
def sortowanie_pozycyjne(dane):
    liczba_cyfr = max([len(str(liczba)) for liczba in dane])

    for i in range(liczba_cyfr):
        sortowanie_przez_zliczanie(dane, i)

    return dane

def sortowanie_przez_zliczanie(dane, pozycja_cyfry):
    lista_zliczen_liczb = [0 for _ in range(10)]
    lista_tymczasowa= [0 for _ in dane]

    podzielnik = 10**pozycja_cyfry

```

```
1
```



Ćwiczenie 3

Napisz program sortujący pozycyjnie, tak aby cyfry na kolejnych pozycjach były sortowane metodą sortowania przez wstawianie. Zbiór ma być posortowany nierosnąco.

Specyfikacja problemu:

Dane:

- `dane` – lista zawierająca liczby całkowite; lista do posortowania

Wynik:

- `posortowane` – lista zawierająca liczby całkowite; posortowana lista wejściowa

Program przetestuj dla listy: `[5000,23,567,34909,2,98,1010,8,90]`

Twoje zadania

1. Program sortuje listę liczb nierosnąco, wykorzystując połączenie algorytmów sortowania pozycyjnego i sortowania przez wstawianie.

```
def sortowanie_przez_wstawianie(dane, pozycja_cyfry):
    podzielnik = 10**pozycja_cyfry
    # Tu uzupełnij kod


def sortowanie_pozycyjne(dane):
    liczba_cyfr = max([len(str(liczba)) for liczba in dane])

    for i in range(liczba_cyfr):
        sortowanie_przez_wstawianie(dane, i)

    return dane

dane = [5000, 23, 567, 34909, 2, 98, 1010, 8, 90]
```

```

```

Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Sortowanie pozycyjne liczb 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:

a) wyszukiwanie elementów liniowe i przez połowienie (do znajdowania elementów w zbiorze, sortowania przez wstawianie, przybliżonego rozwiązywania równań, sprawdzania przynależności punktu do wielokąta wypukłego),

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 implementację algorytmu sortowania pozycyjnego liczb w języku Python.
- Zaimplementujesz w języku Python różne warianty algorytmu sortowania pozycyjnego liczb, wykorzystujące pomocniczo sortowanie kubełkowe, sortowanie bąbelkowe, a także sortowanie przez zliczanie.
- Wykonasz kilka ćwiczeń z programowania dotyczących algorytmu sortowania pozycyjnego.

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 liczb w języku Python". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj" w kontekście programowania.

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć zawarte w sekcji „Wprowadzenie". Następnie wspólnie z uczniami określa kryteria sukcesu.

Faza realizacyjna:

1. **Praca z tekstem.** Nauczyciel ocenia, na podstawie informacji na platformie, stan przygotowania uczniów do zajęć. Jeżeli jest ono niewystarczające prosi o ciche zapoznanie się z treścią w sekcji „Przeczytaj". Uczniowie analizują przedstawione tam przykłady i powtarzają zaprezentowane rozwiązania na swoim komputerze.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Prezentacja multimedialna". Uczniowie wspólnie zapoznają się z jej treścią, zapisując ewentualne problemy i pytania, po czym następuje dyskusja, w trakcie której nauczyciel wyjaśnia niezrozumiałe treści.
3. **Ćwiczenie umiejętności.** Uczniowie realizują indywidualnie ćwiczenia nr 1 i 2 z sekcji „Sprawdź się", po ich wykonaniu porównują otrzymane wyniki z inną osobą.

Faza podsumowująca:

1. Nauczyciel wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji podsumowuje przebieg zajęć, a także wskazuje mocne i słabe strony pracy uczniów.
2. Na koniec zajęć z programowania w Pythonie nauczyciel prosi uczniów o rozwinięcie zdania: „Na dzisiejszych zajęciach nauczyłam/łem się jak…".

Praca domowa:

1. Uczniowie rozwiązują ćwiczenie nr 3 z sekcji „Sprawdź się".
2. Uczniowie zapisują program sortujący pozycyjnie liczby, który wykorzystuje algorytm sortowania przez zliczanie.

Materiały pomocnicze:

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

Wskazówki metodyczne:

- Treści w sekcji „Przeczytaj” można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.