Sortowanie szybkie w języku Python

- Wprowadzenie
- Przeczytaj
- Film samouczek
- Sprawdź się
- Dla nauczyciela

Źródło: Kelvyn Ornettte Sol Marte, domena publiczna.

Wiesz już, że algorytmy różnią się od siebie złożonością czasową. Algorytm sortowania szybkiego, który omówiliśmy w e-materiale Sortowanie szybkie, osiąga zadowalającą złożoność, w związku z czym jest dość powszechnie używany. Sprawdźmy, jak zaimplementować go w języku Python.

Implementację algorytmu sortowania szybkiego w innych językach programowania przedstawiamy w e-materiałach:

- Sortowanie szybkie w języku C++,
- Sortowanie szybkie w języku Java.

Więcej zadań? Przejdź do: Sortowanie szybkie – zadania maturalne.

Informacja na temat złożoności obliczeniowej znajdziesz w e-materiale Złożoność obliczeniowa algorytmów.

> **Twoje cele**
>
> - Powtórzysz wiadomości dotyczące sortowania szybkiego.
> - Zaimplementujesz algorytm sortowania szybkiego w języku Python.
> - Rozwiążesz ćwiczenia wymagające wykorzystania algorytmu sortowania szybkiego.

Przeczytaj

Implementacja algorytmu w języku Python

Algorytm sortowania szybkiego bazuje na dwóch funkcjach. Są to:

1. Funkcja odpowiadająca za podział tablicy – wartością zwracaną przez funkcję będzie indeks miejsca podziału tablicy (indeks pivota). W naszym programie funkcja przyjmie nazwę `podziel_tablice`. Parametry tej funkcji będą identyczne jak w przypadku funkcji `sortowanie_szybkie`.
2. Funkcja wywoływana rekurencyjnie – nazwiemy ją `sortowanie_szybkie`. Funkcja ta będzie przyjmowała trzy parametry:
    - `tab` – tablica zawierająca liczby całkowite do posortowania,
    - `indeks_poczatkowy` – indeks elementu początkowego fragmentu tablicy, od którego chcemy rozpocząć sortowanie tablicy,
    - `indeks_koncowy` – indeks końcowego elementu fragmentu tablicy, na którym chcemy zakończyć sortowanie tablicy.

Oto obie zadeklarowane funkcje:

```
1 def podziel_tablice(tab, indeks_poczatkowy, indeks_koncowy):
2
3 def sortowanie_szybkie(tab, indeks_poczatkowy, indeks_koncowy):
```

Zacznijmy od wypełnienia bloku funkcji `sortowanie_szybkie`. W instrukcji warunkowej sprawdzamy, czy podana jako argument tablica składa się z więcej niż jednego elementu. Jeżeli tak, następuje wywołanie funkcji podziału tablicy – `podziel_tablice`. Do zmiennej `indeks_podzialu` zostanie zapisany punkt podziału tablicy (określony przez funkcję `podziel_tablice`).

Po wywołaniu funkcji podziału następują dwa wywołania rekurencyjne funkcji `sortowanie_szybkie()`:

- `sortowanie_szybkie(tab, indeks_poczatkowy, indeks_podzialu - 1);` – dla pierwszej (lewej) części tablicy,
- `sortowanie_szybkie(tab, indeks_podzialu + 1, indeks_koncowy);` – dla drugiej (prawej) części tablicy.

Oto cały blok funkcji `sortowanie_szybkie()`:

```
def sortowanie_szybkie(tab, indeks_poczatkowy, indeks_koncowy):
    if indeks_poczatkowy < indeks_koncowy:
        indeks_podzialu = podziel_tablice(tab, indeks_poczatkowy,
        sortowanie_szybkie(tab, indeks_poczatkowy, indeks_podzial
        sortowanie_szybkie(tab, indeks_podzialu + 1, indeks_konco
```

Kolejnym krokiem będzie przygotowanie funkcji `podziel_tablice`.

Tworzymy zmienną `pivot`, do której zapisujemy wartość ostatniego elementu z sortowanego zakresu tablicy.

Następnym krokiem jest utworzenie zmiennej `indeks_elementu_mniejszego_od_pivota`, której ustawiamy wartość o 1 mniejszą od tej zapisanej w zmiennej `indeks_poczatkowy`. Będzie to iterator, spełniający dwa ważne zadania. Po pierwsze będzie wskazywał, na którą pozycję mamy przenieść element, gdy okaże się on mniejszy od pivota. Po drugie pomoże ustalić, w którym miejscu znajdują się elementy mniejsze od pivota. Tę zmienną wykorzystamy przy kolejnych podziałach tablicy.

Tworzymy pętlę `for`. Dla czytelności iterator sterujący pętlą określimy jako `indeks_poszukujacy`. Jego zadaniem będzie przejście po wszystkich elementach tablicy `tab`, które znajdują się w zadanym zakresie.

Wewnątrz pętli tworzymy instrukcję warunkową. Jeżeli wartość elementu `tab[indeks_poszukujacy]` jest mniejsza od wartości `pivot`, przenosimy ją na lewą stronę. Zwiększamy `indeks_elementu_mniejszego_od_pivota` i zamieniamy miejscami element znajdujący się w tablicy `tab[indeks_elementu_mniejszego_od_pivot]` z elementem `tab[indeks_poszukujacy]`. Pamiętajmy, że operacja ta jest powtarzana dla każdej komórki w tablicy, której indeks znajduje się w przedziale `<indeks_poczatkowy, indeks_koncowy>`.

Po przejściu pętli porządkujemy pivota – zamieniamy go miejscami z elementem wskazywanym przez `indeks_elementu_mniejszego_od_pivota + 1`.

Funkcję kończymy, zwracając indeks wskazujący na miejsce podziału: `indeks_elementu_mniejszego_od_pivota + 1`.

```
def podziel_tablice(tab, indeks_poczatkowy, indeks_koncowy):
    pivot = tab[indeks_koncowy]
    indeksElementuMniejszegoOdPivota = indeks_poczatkowy - 1

    for indeksPoszukujacy in range(indeks_poczatkowy, indeks_konc
        if tab[indeksPoszukujacy] < pivot:
```

```
            indeksElementuMniejszegoOdPivota += 1
            tab[indeksElementuMniejszegoOdPivota], tab[indeksPosz

    tab[indeksElementuMniejszegoOdPivota + 1], tab[indeks_koncowy

    return indeksElementuMniejszegoOdPivota + 1
```

Utworzymy tablicę z liczbami całkowitymi, a następnie wywołamy funkcję `sortowanie_szybkie` w celu posortowania elementów. Wypiszemy też elementy tablicy przed sortowaniem i po sortowaniu, żeby zobaczyć, czy zostało ono wykonane poprawnie.

```
tab = [4, 5, 65, 7, 4, 0, 23, 34, 12, 1, 3, 4, 5]
ostatni_element = len(tab)

for i in range(ostatni_element):
    print(tab[i], end="  ")

print()

sortowanie_szybkie(tab, 0, len(tab) - 1)

for i in range(ostatni_element):
    print(tab[i], end="  ")
```

Oto pełen kod programu:

```
def sortowanie_szybkie(tab, indeks_poczatkowy, indeks_koncowy):
    if indeks_poczatkowy < indeks_koncowy:
        indeks_podzialu = podziel_tablice(tab, indeks_poczatkowy,
        sortowanie_szybkie(tab, indeks_poczatkowy, indeks_podzial
        sortowanie_szybkie(tab, indeks_podzialu + 1, indeks_konco

def podziel_tablice(tab, indeks_poczatkowy, indeks_koncowy):
    pivot = tab[indeks_koncowy]
    indeksElementuMniejszegoOdPivota = indeks_poczatkowy - 1

    for indeksPoszukujacy in range(indeks_poczatkowy, indeks_konc
        if tab[indeksPoszukujacy] < pivot:
            indeksElementuMniejszegoOdPivota += 1
            tab[indeksElementuMniejszegoOdPivota], tab[indeksPosz
```

```
15
16      tab[indeksElementuMniejszegoOdPivota + 1], tab[indeks_koncowy
17
18      return indeksElementuMniejszegoOdPivota + 1
19
20  tab = [4, 5, 65, 7, 4, 0, 23, 34, 12, 1, 3, 4, 5]
21  ostatni_element = len(tab)
22
23  for i in range(ostatni_element):
24      print(tab[i], end="  ")
25
26  print()
27
28  sortowanie_szybkie(tab, 0, len(tab) - 1)
29
30  for i in range(ostatni_element):
31      print(tab[i], end="  ")
```

> **Dla zainteresowanych**
>
> W języku Python istnieje wbudowana funkcja `sorted(list)` lub metoda `list.sort()`, która działa w oparciu o zmodyfikowany algorytm timesort. Więcej o sortowaniu w języku Python można znaleźć na stronie Sorting HOW TO.

Słownik

> **pivot**
>
> element osiowy – element tablicy wybrany wedle ustalonego schematu; jest to element odpowiadający za sposób dzielenia tablicy
> **rekurencja**
>
> proces polegający na wywoływaniu funkcji przez siebie samą do momentu rozwiązania określonego problemu
> **wywołanie rekurencyjne**
>
> sytuacja, w której funkcja wywołuje samą siebie
> **zasada „dziel i zwyciężaj”**
>
> zasada często stosowana przez programistów przy tworzeniu algorytmów opartych o rekurencję, polegająca na tym, że jeden trudny, złożony problem dzielony jest na kilka mniejszych, prostszych do rozwiązania
> **złożoność czasowa**

> ilość czasu potrzebnego do wykonania zadania, wyrażona jako funkcja ilości danych
> **złożoność obliczeniowa**
>
> złożoność czasowa i złożoność pamięciowa; ilość zasobów komputerowych potrzebnych do wykonania zadania

Film samouczek

Polecenie 1

W lutym 2000 CBOS przeprowadziło pewien sondaż, ramach którego zadano ankietowanym pytanie: „Czy pana/pani zdaniem równość w społeczeństwie powinna, czy też nie powinna oznaczać, że wyrównany jest materialny poziom życia obywateli?”.

„Zdecydowanie tak” odpowiedziało 30% pytanych; „Raczej tak” – 26%; „Raczej nie” – 24%, „Zdecydowanie nie” – 15%, „Trudno powiedzieć” – 5%.

Napisz program sortujący niemalejąco wyniki ankiety, wykorzystując sortowanie szybkie (***quick sort***).

Specyfikacja problemu:

Dane:

- `liczba_elementow` – liczba odpowiedzi w ankiecie; liczba naturalna
- `dane` – tablica o rozmiarze `liczba_elementow` zawierająca liczby całkowite mniejsze od 100, których suma wynosi 100

Wynik:

- tablica zawierająca dane z ankiety posortowana niemalejąco

Polecenie 2

Porównaj swoje rozwiązanie z przedstawionym w filmie.

Film dostępny pod adresem /preview/resource/R1JePwel6hvxq

Film przedstawia etapy pisania programu w języku Python, sortującego wyniki ankiety dotyczącej równości w życiu społecznym.

Plik o rozmiarze 1.22 KB w języku polskim

> **Ważne!**
>
> Operacja *swap*, o której mowa w filmie, polega na zamianie miejscami dwóch elementów; w tym przypadku są to dwa elementy tablicy.
> **Ważne!**
>
> Mediana to wartość środkowa. Aby wyznaczyć medianę jakiegoś zbioru liczb, musimy najpierw wypisać te liczby w kolejności niemalejącej, a następnie wybrać liczbę środkową (w przypadku, gdy mamy nieparzystą liczbę liczb w zbiorze). Jeśli mamy parzystą liczbę liczb w zbiorze, to mediana jest równa średniej arytmetycznej dwóch środkowych liczb.

Sprawdź się

Pokaż ćwiczenia: 

Pokaż ćwiczenia: 

Ćwiczenie 1

Napisz program, który wykorzystując algorytm sortowania szybkiego, wypisze wartość minimalną oraz maksymalną z podanej tablicy. Przetestuj działanie programu dla tablicy `tab = [9, 11, 0, -8, 11, 5, 20, 45, 0, 100]`.

Specyfikacja problemu:

Dane:

- n – liczba naturalna; liczba elementów tablicy `tab`
- `tab` – n-elementowa tablica liczb całkowitych

Wynik:

- minimum i maksimum dla tablicy `tab`; liczby całkowite

Przykładowe wyjście:

```
-8 100
```

Twoje zadania

1. Program powinien wyświetlić dwie liczby oddzielone spacją.

Ćwiczenie 2

Napisz program, który przy użyciu algorytmu sortowania szybkiego uporządkuje zbiór podanych liter alfabetu łacińskiego w kolejności odwrotnej do alfabetycznej. Przetestuj działanie programu dla następującego zbioru liter `tab = [a, f, e, o, b, l, q, y]`.

Specyfikacja problemu:

Dane:

- `n` – liczba naturalna; liczba elementów tablicy `tab`
- `tab` – n-elementowa tablica zawierająca małe litery alfabetu łacińskiego

Wynik:

- `tab` – tablica znaków posortowana w kolejności odwrotnej do kolejności alfabetycznej; elementy oddzielone są pojedynczym znakiem spacji

Przykładowe wyjście:

```
y q o l f e b a
```

Twoje zadania

1. Program powinien wyświetlić tablicę posortowaną odwrotnie do kolejności alfabetycznej.

Ćwiczenie 3

W ramach badania zapytano grupę respondentów o zarobki. Odpowiedzi umieszczono w tablicy. Użyj algorytmu sortowania szybkiego, aby znaleźć **medianę** zarobków w tej grupie. Program powinien wydrukować wynik na standardowe wyjście.

> **Ważne!**
>
> Mediana to wartość środkowa. Aby wyznaczyć medianę jakiegoś zbioru liczb, musimy najpierw wypisać te liczby w kolejności niemalejącej, a następnie wybrać liczbę środkową (w przypadku, gdy mamy nieparzystą liczbę liczb w zbiorze). Jeśli mamy parzystą liczbę liczb w zbiorze, to mediana jest równa średniej arytmetycznej dwóch środkowych liczb.

Przetestuj jego działanie dla tablicy
`zarobki = [8500.57, 6400.32, 2800.56, 3500.12, 12870.67, 3300.45, 7020.0, 3000.01, 8100.29].`

Specyfikacja problemu:

Dane:

- n – liczba naturalna; liczba elementów tablicy `zarobki`
- `zarobki` – n-elementowa tablica liczb rzeczywistych

Wynik:

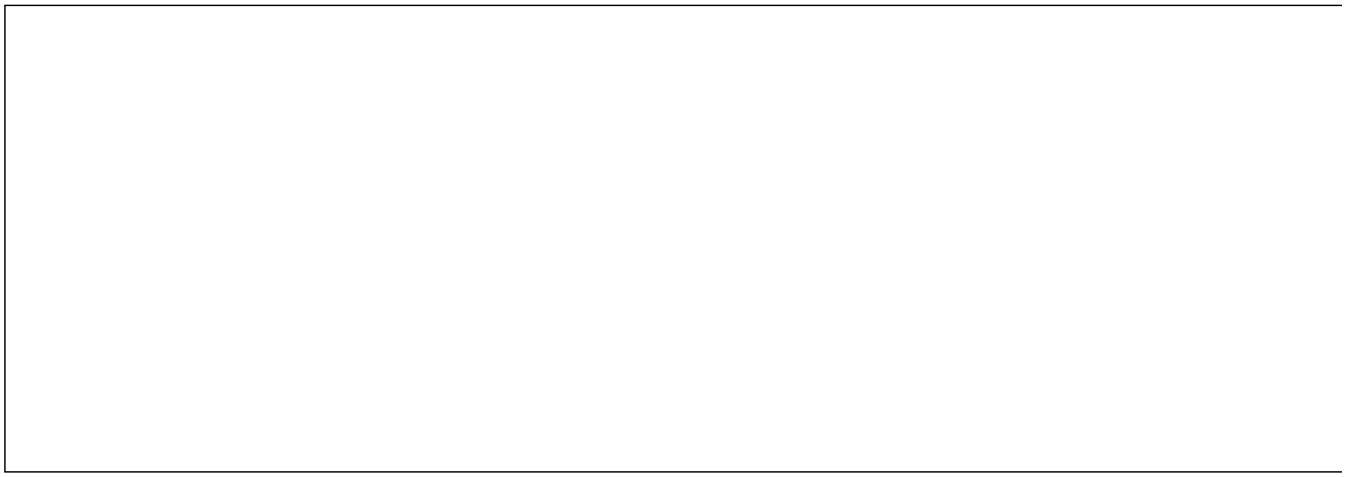
- mediana posortowanej tablicy `zarobki`; liczba rzeczywista

Przykładowe wyjście

```
6400.32
```

Twoje zadania

1. Program powinien wyświetlić jedną liczbę – medianę wartości zbioru. Przykład: dane wejściowe 5 3 4 1 3, wynik: 3.



Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Sortowanie szybkie 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 + 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:

c) metodę dziel i zwyciężaj (jednoczesne znajdowanie minimum i maksimum, sortowanie przez scalanie i szybkie),

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):

- Powtórzysz wiadomości dotyczące sortowania szybkiego.

- Zaimplementujesz algorytm sortowania szybkiego w języku Python.
- Rozwiążesz ćwiczenia wymagające wykorzystania algorytmu sortowania szybkiego.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- metody aktywizujące.

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;
- telefony z dostępem do internetu;
- 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 szybkie w języku Python". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj" w kontekście programowania.

Faza wstępna:

1. Nauczyciel wprowadza uczniów szczegółowo w temat lekcji i jej cele. Może posłużyć się wyświetloną na tablicy zawartością sekcji „Wprowadzenie".
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel prosi wybranego ucznia lub uczniów o przedstawienie sytuacji problemowej związanej z tematem lekcji oraz w odniesieniu do poznanych języków programowania.

Faza realizacyjna:

1. **Praca z tekstem.** Nauczyciel wyświetla zawartość sekcji „Przeczytaj". Uczniowie wspólnie zapoznają się z jego treścią. Następnie indywidualnie wykonują przedstawiony program.
2. **Praca z multimedium**. 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 „Film samouczek". Uczniowie analizują przedstawiony tam przykład i powtarzają zaprezentowane rozwiązanie na swoim komputerze.
3. **Ćwiczenie umiejętności.** Poszukiwanie najefektywniejszego rozwiązania problemu. Uczniowie wykonują w parach ćwiczenie nr 2 z sekcji „Sprawdź się", a następnie porównują swój kod omawiając go wspólnie na forum. Nauczyciel ocenia efektywność zastosowanego rozwiązania.

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat lekcji zawarty w sekcji „Wprowadzenie" i inicjuje krótką rozmowę na temat zrealizowanych celów (czego uczniowie się nauczyli).

Praca domowa:

1. Uczniowie wykonują ćwiczenie 3 z sekcji „Sprawdź się".

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.