Znajdowanie określonego elementu w zbiorze w języku Python

Źródło: Alex Block, domena publiczna.

Współczesne komputery przetwarzają ogromne ilości danych – kursy walut, ceny akcji na giełdzie, dane meteorologiczne itd. Informacje te byłyby jednak bezużyteczne, gdybyśmy nie mogli zweryfikować wystąpienia pewnych danych, np. sprawdzić, przez ile dni akcje spółki kosztowały mniej niż 50 zł lub czy w maju wystąpił jakikolwiek dzień, w którym temperatura w Polsce spadła poniżej zera. Aby uzyskać odpowiedzi na tego typu pytania, możemy wykorzystać algorytmy wyszukiwania.

W tym e-materiale poznasz implementację algorytmu wyszukiwania określonego elementu w zbiorze w języku Python. W e-materiale Znajdowanie określonego elementu w zbiorze znajdziesz ogólne informacje dotyczące szukania idola oraz lidera w zbiorze.

Implementacja w innych językach programowania:

- Znajdowanie określonego elementu w zbiorze w języku C++,
- Znajdowanie określonego elementu w zbiorze w języku Java.

Więcej zadań? Znajdowanie określonego elementu w zbiorze – zadania maturalne

Twoje cele

- Wyszukasz idola w języku Python.
- Przeanalizujesz działanie algorytmu wyszukiwania binarnego w języku Python.

- Rozwiążesz zadanie w języku Python, związane z wyszukiwaniem danych.

Przeczytaj

Lider w zbiorze

Liderem nazywamy liczbę, która w zbiorze liczącym n elementów występuje więcej niż $\frac{n}{2}$ razy. Ponieważ inna liczba nie może występować w tym samym zbiorze liczbowym więcej niż $\frac{n}{2}$ razy, lider jest tylko jeden. W zbiorze {1, 4, 6, 1, 1} liderem jest liczba 1 – występuje w nim aż 3 razy.

$$\frac{n}{2} = \frac{5}{2} = 2,5 < 3$$

Natomiast w zbiorze {8, 6, 4, 5, 3, 2} żadna z sześciu liczb nie jest liderem.

Właściwości zbiorów z liderem

Zbiory, które mają lidera, posiadają bardzo ważną właściwość: jeżeli z takiego zbioru usuniemy parę liczb – pod warunkiem, że będą one różne – zbiór ten nadal będzie miał tego samego lidera. Przeanalizujmy dwa przykłady:

- jeden, w którym obie usunięte liczby ze zbioru nie są liderem;
- drugi, w którym jedna z liczb jest liderem, natomiast druga nie.

Ze zbioru {1, 4, 6, 1, 1} usuwamy liczby 4 oraz 6 i zostajemy z podzbiorem {1, 1, 1}. Ponieważ zbiór ten składa się tylko z jedynek, bardzo łatwo stwierdzić, iż lider tego zbioru nie zmienił się i nadal jest nim liczba 1.

W drugim przypadku usuniemy jedną liczbę, która nie jest liderem, i jedną, która nim jest – usuniemy liczby: 4 oraz 1. Zostajemy wtedy ze zbiorem {1, 6, 1}, którego liderem nadal jest liczba 1.

Ponieważ para, która zostaje wyeliminowana, nie może składać się z tych samych liczb, przypadek, w którym usunięte zostaną dwie liczby lidera, nie zaistnieje.

Jeżeli weźmiemy jakikolwiek zbiór i będziemy odejmować od niego po dwa różne elementy, aż do momentu, w którym nie będziemy mogli usunąć już więcej, otrzymamy jeden z trzech możliwych rezultatów:

- pusty zbiór – oznacza, że zbiór nie ma lidera,
- zbiór z jednym elementem – jeżeli zbiór ma lidera, to będzie nim właśnie ten element, a jeżeli nie ma, to jest to po prostu element, który nie został usunięty; żeby upewnić się,

czy ten element jest liderem, należy policzyć, ile razy wystąpił on w oryginalnym zbiorze i sprawdzić, czy liczba ta jest większa niż $\frac{n}{2}$,
- zbiór zawiera więcej niż jeden element, ale wszystkie pozostałe elementy mają takie same wartości – w tym przypadku element, który pozostał, na pewno jest liderem.

Algorytm wyszukiwania lidera

W e-materiale teoretycznym poznaliśmy algorytm wyszukiwania lidera w zbiorze. Jego implementacja w języku Python wygląda następująco:

```
zbior = [1, 3, 1]
lider = zbior[0]
n = 1

for i in range(1,len(zbior)):
    if n > 0:
        if lider == zbior[i]:
            n = n + 1
        else:
            n = n - 1
    else:
        lider = zbior[i]
        n = 1

if n == 0:
    print("Zbiór nie ma lidera")
    exit()

liczba_wystapien_lidera = 0
for i in  range(0,len(zbior)):
    if zbior[i] == lider:
        liczba_wystapien_lidera += 1

if liczba_wystapien_lidera > len(zbior) / 2:
    print("Liderem zbioru jest ",lider)
else:
    print("Zbiór nie ma lidera")
```

Wyszukiwanie idola w zbiorze

Wyobraźmy sobie pewną grupę, w której każda osoba może, ale nie musi, znać pozostałe osoby. Ponieważ to, że osoba A zna osobę B, nie znaczy, że osoba B zna osobę A, może dojść do specyficznej sytuacji, w której odnajdziemy jedną osobę znaną przez wszystkich, jednocześnie nieznającą nikogo. Taką osobę nazywamy idolem.

To, czy dana osoba zna inną, przedstawimy za pomocą tzw. macierzy sąsiedztwa. Będzie to dwuwymiarowa tablica wypełniona zerami, jeżeli relacja znajomości z drugą osobą nie zachodzi, oraz jedynkami, jeżeli ta relacja zaistniała. Każdy wiersz tej macierzy będzie oznaczał stan wiedzy jednej osoby, zaś każda kolumna będzie oznaczała wiedzę o konkretnej osobie. Uznajemy, że żadna osoba nie zna samej siebie, dlatego przekątna tej tablicy będzie wypełniona zerami.

```
macierz = [
    [0, 1, 0, 1, 1],
    [0, 0, 1, 0, 1],
    [0, 1, 0, 1, 1],
    [1, 1, 1, 0, 1],
    [0, 0, 0, 0, 0]
]
```

Teraz zapiszemy pętlę, która przejdzie po wszystkich wierszach tej macierzy. W każdym jej wykonaniu będziemy sprawdzali, czy znaleźliśmy wiersz wypełniony samymi zerami – oznacza to, że mamy potencjalnego idola. Jeżeli okaże się, że dowolna inna osoba go nie zna, zbiór nie będzie miał innego idola.

```
macierz = [
    [0, 1, 0, 1, 1],
    [0, 0, 1, 0, 1],
    [0, 1, 0, 1, 1],
    [1, 1, 1, 0, 1],
    [0, 0, 0, 0, 0]
]

for i in range(len(macierz)):
    if macierz[i] == [0]*len(macierz):
```

Teraz stworzymy kolejną pętlę, która przejdzie po wszystkich elementach w **kolumnie** informującej nas o tym, czy osoba o indeksie `i` jest znana. Wyjątkiem będzie tutaj sam wiersz wypełniony zerami – zostanie on pominięty. Jeżeli znajdziemy tam

jakiekolwiek zero, natychmiastowo stwierdzimy, że zbiór ten nie ma idola i zakończymy wykonywanie pętli.

```
macierz = [
    [0, 1, 0, 1, 1],
    [0, 0, 1, 0, 1],
    [0, 1, 0, 1, 1],
    [1, 1, 1, 0, 1],
    [0, 0, 0, 0, 0]
]

for i in range(len(macierz)):
    if macierz[i] == [0]*len(macierz):
        for x in macierz[:i] + macierz[i+1:]:
            if x[i] == 0:
                print("Zbior nie ma idola")
                break
```

Następnie, ponownie korzystając z konstrukcji `for-else`, dodamy wyświetlanie informacji o znalezionym idolu. Jeżeli pętla przechodząca po wszystkich elementach w kolumnie wykona się bez wywołania instrukcji `break`, wtedy mamy pewność, że osoba o indeksie `i` jest idolem. Dodamy też instrukcję `else` na koniec pętli przechodzącej po wierszach – odpowiada to sytuacji, w której nie znaleźliśmy żadnej osoby nieznającej nikogo.

```
macierz = [
    [0, 1, 0, 1, 1],
    [0, 0, 1, 0, 1],
    [0, 1, 0, 1, 1],
    [1, 1, 1, 0, 1],
    [0, 0, 0, 0, 0]
]

for i in range(len(macierz)):
    if macierz[i] == [0]*len(macierz):
        for x in macierz[:i] + macierz[i+1:]:
            if x[i] == 0:
                print("Zbior nie ma idola")
                break
        else:
            print(f"Idol to {i}")
```

```
        break
else:
    print("Zbior nie ma idola")
```

Po uruchomieniu tego programu otrzymamy informację, że idolem jest osoba o numerze 4.

Przykładowe macierze, dla których idol nie istnieje:

```
macierz = [
    [0, 1, 0, 1, 1],
    [0, 0, 1, 0, 1],
    [0, 1, 0, 1, 1],
    [1, 1, 1, 0, 0],
    [0, 0, 0, 0, 0]
]
```

```
macierz = [
    [0, 1, 0, 1, 1],
    [0, 0, 1, 0, 1],
    [0, 1, 0, 1, 1],
    [1, 1, 1, 0, 0],
    [0, 0, 0, 1, 0]
]
```

Słownik

> **macierz sąsiedztwa**
>
> w teorii grafów jest to tablica zer i jedynek, informująca o tym, czy istnieją ścieżki między odpowiednimi wierzchołkami grafu
> **zbiór uporządkowany**
>
> zbiór, w którym wszystkie elementy są posortowane według pewnej ich cechy; w przypadku liczb najczęściej jest to ich wartość

Film samouczek

Polecenie 1

Napisz program, który sprawdzi, czy uporządkowany zbiór `dane` zawiera element o wartości `szukany_element`. Swoje rozwiązanie przetestuj dla pięcioelementowego zbioru `{2, 4, 6, 8, 10}` oraz `szukany_element = 4`.

Specyfikacja problemu:

Dane:

- `dane` – zbiór liczb całkowitych
- `szukany_elementy` – element szukany w zbiorze

Wynik:

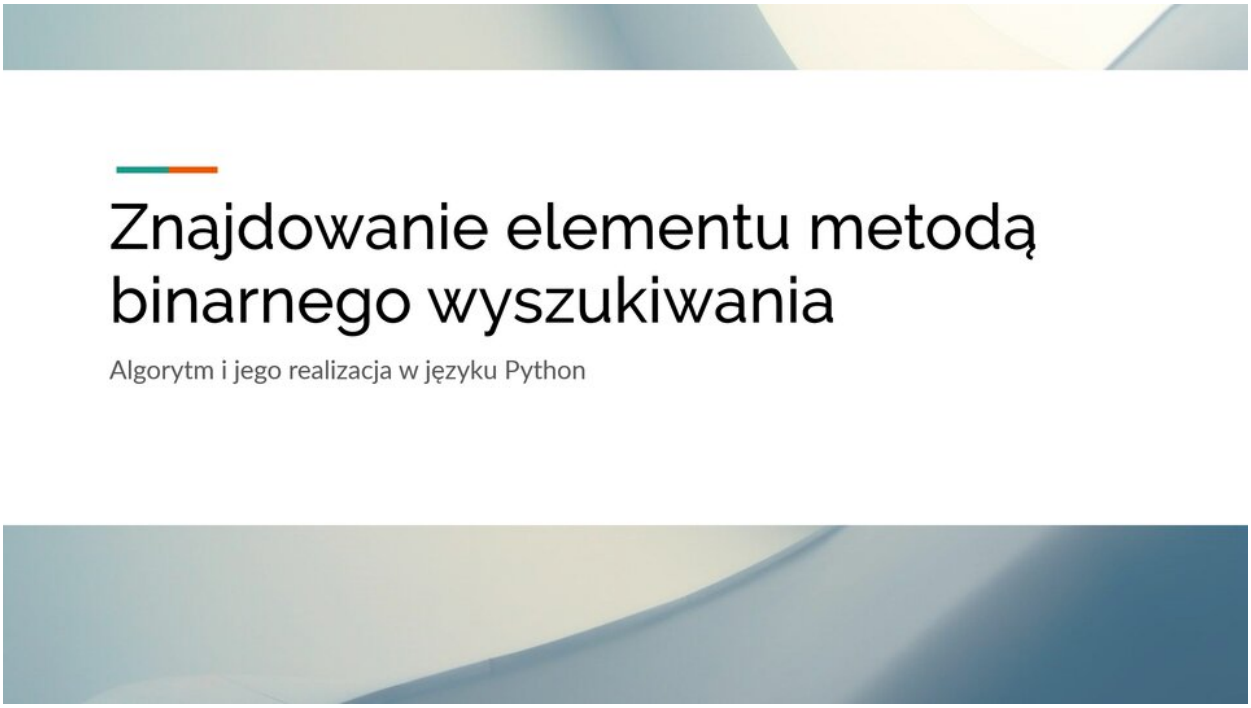
Na konsoli wyświetli się indeks szukanego elementu (komunikat `Znaleziono szukany element pod indeksem: [indeks]`) lub komunikat o braku elementu w zbiorze (komunikat `Nie znaleziono szukanego elementu`).

```
# Tutaj dodaj własny kod. Użyj funkcji
# print()
```

```
```

Polecenie 2

Porównaj swoje rozwiązanie z tym przedstawionym w filmie.



Film dostępny pod adresem /preview/resource/RMupoX4lSIkz0

Film nawiązujący do treści materiału: Znajdowanie elementu metodą binarnego wyszukiwania.

Plik o rozmiarze 784.00 B w języku polskim

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Napisz program, który wypisze, ile razy w liście `zbior` występuje liczba `x`.

Działanie programu przetestuj dla wartości `x = 21`.

Specyfikacja problemu:

Dane:

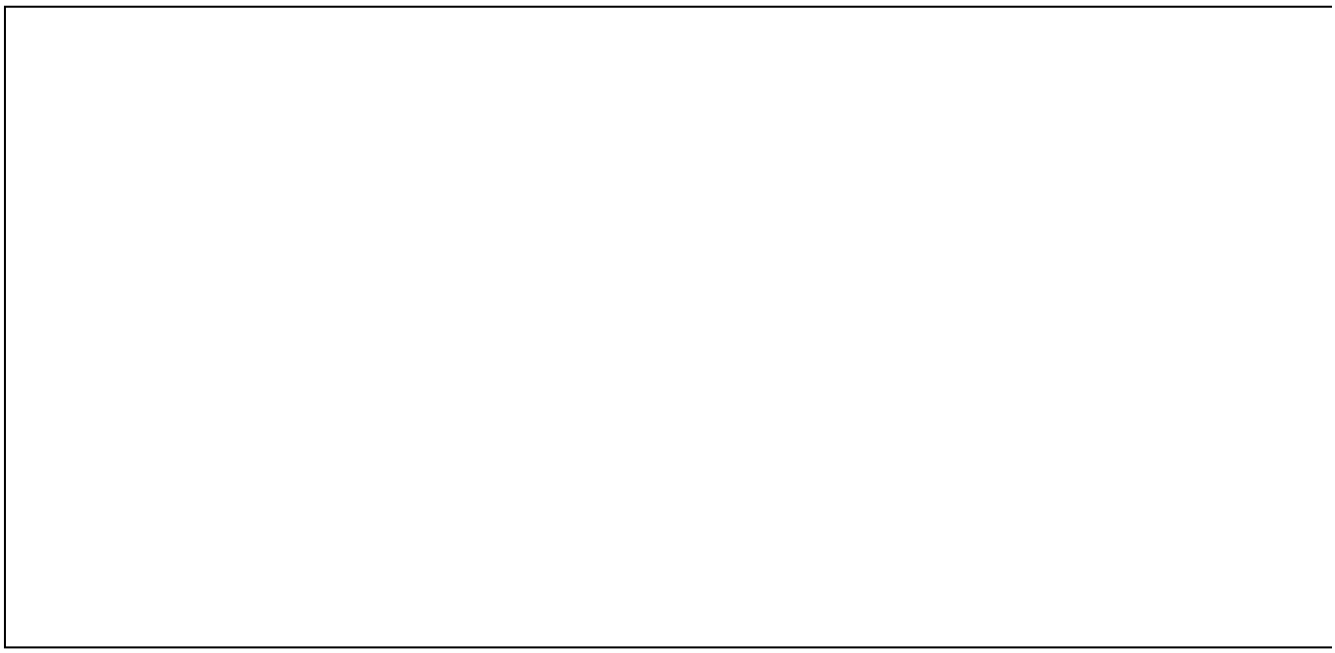
- `zbior` – lista liczb całkowitych

Wynik:

- `licznik` – liczba całkowita

Twoje zadania

1. Program wypisuje, ile razy liczba 21 pojawia się w liście `zbior`.



Ćwiczenie 2

Napisz program znajdujący lidera w zbiorze.

Działanie programu przetestuj dla zbioru
`zbior = [2, 4, 2, 3, 5, 6, 2, 2, 8, 2, 2, 2]`.

Specyfikacja problemu:

Dane:

- `zbior` – lista liczb całkowitych

Wynik:

- `lider` – liczba całkowita

Twoje zadania

1. Program znajduje lidera zbioru.

Ćwiczenie 3

Napisz program wykorzystujący algorytm wyszukiwania binarnego, który wypisze indeks wybranej liczby ze zbioru `zbior` lub poda komunikat, że liczba nie istnieje.

Działanie programu przetestuj dla liczby 24.

Specyfikacja problemu:

Dane:

- `zbior` – lista liczb całkowitych
- `szukanaLiczba` – liczba całkowita

Wynik:

- `lewaGranica` – liczba całkowita

Twoje zadania

1. Program znajduje indeks liczby w zbiorze.

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Znajdowanie określonego elementu w zbiorze 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:

1) zapisuje za pomocą listy kroków, schematu blokowego lub pseudokodu, i implementuje w wybranym języku programowania, algorytmy poznane na wcześniejszych etapach oraz algorytmy:

b) znajdowania określonego elementu w zbiorze: lidera, idola, elementu w zbiorze uporządkowanym metodą binarnego wyszukiwania,

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):

- Wyszukasz lidera w języku Python.

- Przeanalizujesz działanie algorytmu wyszukiwania binarnego w języku Python.
- Rozwiążesz zadanie w języku Python, związane z wyszukiwaniem danych.

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: „Znajdowanie określonego elementu w zbiorze w języku Python". Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Prowadzący wyświetla na tablicy interaktywnej zawartość sekcji „Wprowadzenie" i omawia cele do osiągnięcia w trakcie lekcji.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel prosi wybranego ucznia lub uczniów o przedstawienie sytuacji problemowej związanej z tematem lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Uczniowie przystępują do cichego czytania tekstu zawartego w sekcji „Przeczytaj", jeśli nauczyciel - na podstawie raportu na platformie - uważa, że

przygotowanie uczniów jest wystarczające, może pominąć tę czynność.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Film samouczek". Uczniowie wspólnie analizują polecenie 1. W parach piszą program, a następnie porównują swoje rozwiązanie z przedstawionym w filmie.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenie nr 1 z sekcji „Sprawdź się", a następnie porównują swoje odpowiedzi z kolegą lub koleżanką.
4. Liga zadaniowa – uczniowie pracując w parach, wykonują ćwiczenie nr 2 z sekcji „Sprawdź się", a następnie dzielą się swoimi wynikami przez porównywanie napisanego kodu z inną grupą, która również zakończyła zadanie.

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. Nauczyciel prosi uczniów o podsumowanie zgromadzonej wiedzy w zakresie programowania w języku Python.

Praca domowa:

1. Uczniowie wykonują ćwiczenie nr 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:

- Uczniowie mogą wykorzystać multimedium w sekcji „Film samouczek" do przygotowania się do lekcji powtórkowej.