Faktoryzacja w języku Python

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

Źródło: Franck V., domena publiczna.

Rozkładanie liczby na czynniki pierwsze jest istotnym elementem kryptografii. Własności tego procesu określają, jak trudny do złamania będzie używany przez nas szyfr. Faktoryzacja małej liczby nie stanowi problemu, natomiast rozkład dużych liczb na czynniki pierwsze jest bardzo wymagający obliczeniowo.

W e-materiale Faktoryzacja poznaliśmy algorytm rozkładu liczb na czynniki pierwsze. W tym e-materiale dowiemy się, jak wykorzystać go w programach napisanych w języku Python.

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

- Faktoryzacja w języku C++,
- Faktoryzacja w języku Java.

Więcej zadań? Sięgnij do Faktoryzacja – zadania maturalne.

> **Twoje cele**
>
> - Przeanalizujesz algorytm rozkładu liczby na czynniki pierwsze.
> - Zaimplementujesz algorytm rozkładu liczby na czynniki pierwsze.
> - Przetestujesz program dla różnych danych.

Film samouczek

Polecenie 1
Napisz program, który dla podanej przez użytkownika liczby naturalnej n wypisze jej czynniki pierwsze.

Specyfikacja problemu:

Dane:

- n – liczba naturalna; $n > 1$

Wynik:

Program na standardowym wyjściu wypisze czynniki pierwsze liczby n.

Polecenie 2
Porównaj swoje rozwiązanie z przedstawionym w filmie.

Trwa wczytywanie danych ..

Film dostępny pod adresem /preview/resource/R1M3jVunhUkFS

Film nawiązujący do rozkładu liczb pierwszych - implementacji algorytmu faktoryzacji w języku Python.

Plik o rozmiarze 242.00 B w języku polskim

Plik o rozmiarze 340.00 B w języku polskim

Polecenie 3

Przeczytaj

Porównanie sposobów rozkładu liczby na czynniki pierwsze

W filmie z sekcji „Film samouczek” przedstawiliśmy zoptymalizowany sposób rozkładu liczby na czynniki pierwsze.

Spróbujmy zapisać funkcję, która będzie wykonywać rozkład dowolnej liczby na czynniki pierwsze z jednoczesnym zliczaniem potencjalnych dzielników i obliczaniem czasu wykonania tych obliczeń. Skorzystamy w tym celu z funkcji `time()`.

```
def rozklad(Q):
    from math import sqrt
    from time import time
    t0 = time()

    # górna granica sprawdzanych dzielników
    g = int(sqrt(Q))+1

    # tablica czynników pierwszych
    tab = []

    # licznik sprawdzonych dzielników
    dzielniki = 0

    for p in range(2,g):
        dzielniki += 1
        while Q % p == 0:
            tab.append(p)
            Q = Q // p
            if Q == 1:
                break
        if Q == 1:
            break

    if Q > 1:
        tab.append(Q)

    t1 = time()
    print(f'Znaleziono czynniki pierwsze: {tab}')
```

```
    print(f'Czas w sekundach: {t1-t0:.6f}')
    print(f'Sprawdzone dzielniki {dzielniki}')
```

Możemy przetestować program dla kilku liczb:

```
rozklad(44100)
# Znaleziono czynniki pierwsze: [2, 2, 3, 3, 5, 5, 7, 7]
# Czas w sekundach: 0.000015
# Sprawdzone dzielniki: 6

rozklad(44103241230)
# Znaleziono czynniki pierwsze: [2, 3, 5, 97, 587, 25819]
# Czas w sekundach: 0.003861
# Sprawdzone dzielniki: 25818

rozklad(372036854775807)
# Znaleziono czynniki pierwsze: [3, 3, 13, 3179802177571]
# Czas w sekundach: 2.584548
# Sprawdzone dzielniki: 19288255

rozklad(133720368547758O7)
# Znaleziono czynniki pierwsze: [13, 19, 271, 439, 5849, 77801]
# Czas w sekundach: 0.007574
# Sprawdzone dzielniki: 77800

rozklad(4133720368547758O7)
# Znaleziono czynniki pierwsze: [1051, 15263, 25769053939]
# Czas w sekundach: 87.726839
# Sprawdzone dzielniki: 642940149
```

Czas wykonywania obliczeń dla liczb, które mają duże czynniki pierwsze, wyniósł prawie 1,5 minuty. Sprawdzanie kolejnych liczb dopóki, dopóty znaleziony zostanie pierwiastek, nie jest najefektywniejszym możliwym sposobem. Jedną z lepszych metod przedstawimy poniżej.

Drugi sposób rozkładu liczby na czynniki pierwsze

Dla każdej liczby `Q` możemy określić wszystkie jej dzielniki, sprawdzając jej podzielność przez kolejne liczby pierwsze, a więc 2, 3 oraz liczby postaci:

$$p = 6 \cdot k \pm 1$$

gdzie:

$$k = 1, 2, \ldots, g$$

$$g = \sqrt{Q}$$

Za pomocą zmiennej d będziemy w stanie sprawdzić wszystkie dzielniki w jednej pętli (najpierw obliczymy $6 \cdot k - 1$, a następnie $6 \cdot k + 1$ i zwiększymy k).

```
def rozklad_2(Q):
    from math import sqrt
    from time import time
    t0 = time()

    # granice sprawdzanych dzielników
    p = 2
    g = int(sqrt(Q))

    # zmienne do wyliczania p = 6 * k +/- 1
    k = 1
    d = -1

    # licznik sprawdzonych dzielników
    dzielniki = 0

    # tablica czynników pierwszych
    tab = []

    while p <= g:
        dzielniki += 1
        while Q % p == 0:
            tab.append(p)
            Q //= p
        if Q == 1:
            break
        if p < 3:
            p += 1
        else:
            p = 6*k + d
            if d == 1:
```

```
                d = -1
                k += 1
            else:
                d = 1

    if Q > 1:
        tab.append(Q)

    t1 = time()
    print(f'Znaleziono czynniki pierwsze: {tab}')
    print(f'Czas w sekundach: {t1-t0:.6f}')
    print(f'Sprawdzone dzielniki: {dzielniki}')
```

```
rozklad_2(44100)
# Znaleziono czynniki pierwsze: [2, 2, 3, 3, 5, 5, 7, 7]
# Czas w sekundach: 0.000014
# Sprawdzone dzielniki: 4

rozklad_2(44103241230)
# Znaleziono czynniki pierwsze: [2, 3, 5, 97, 587, 25819]
# Czas w sekundach: 0.003591
# Sprawdzone dzielniki: 8608

rozklad_2(372036854775807)
# Znaleziono czynniki pierwsze: [3, 3, 13, 3179802177571]
# Czas w sekundach: 1.642628
# Sprawdzone dzielniki: 6429420

rozklad_2(13372036854775807)
# Znaleziono czynniki pierwsze: [13, 19, 271, 439, 5849, 77801]
# Czas w sekundach: 0.005434
# Sprawdzone dzielniki: 25935

rozklad_2(4133720368547758O7)
# Znaleziono czynniki pierwsze: [1051, 15263, 25769053939]
# Czas w sekundach: 58.880058
# Sprawdzone dzielniki: 214313384
```

Jak widać, druga wersja rozkładu liczby na czynniki pierwsze jest dużo szybsza. Dla największej z testowanych liczb pierwszych czas obliczeń był o ponad 30% krótszy

w stosunku do pierwszego sposobu. Sprawdzono również niemal trzy razy mniej dzielników.

> **Dla zainteresowanych**
>
> Algorytm szyfrowania RSA bazuje na wyniku mnożenia dwóch dużych liczb pierwszych. Aby go złamać, trzeba znaleźć dla danej liczby Q takie dwie różne liczby `e` oraz `t`, których iloczyn wynosi właśnie `Q`. Będzie wówczas można nazwać `t` oraz `e` **faktorami** liczby `Q`.
>
> Przykładowo:
>
> $$4\,295\,229\,443 \;=\; 65\,537 \cdot 65\,539$$

Słownik

> **faktoryzacja liczby Q**
>
> proces, w którym dla liczby $Q > 1$ odnajdujemy takie liczby naturalne $f_1, f_2, f_3 \ldots$, różne od 1 oraz różne od Q (nietrywialne czynniki rozkładu), których iloczyn $f_1 * f_2 * f_3 * \ldots = Q$; liczby te nazywamy faktorami

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Napisz program, który wypisze czynniki pierwsze liczby naturalnej dodatniej n. Sprawdź jego działanie dla n = 18.

Specyfikacja problemu:

Dane:

- n – liczba naturalna dodatnia; n > 1

Wynik:

Program, na wyjściu standardowym, wypisze czynniki pierwsze liczby n.

Ćwiczenie 2

Napisz program rozkładający liczbę naturalną dodatnią n na czynniki pierwsze, a następnie wyświetlający na ekranie największy czynnik, jaki pojawił się w rozkładzie. Sprawdź działanie programu dla $n = 5625$.

Specyfikacja problemu:

Dane:

- n – liczba naturalna dodatnia; $n > 1$

Wynik:

Program, na wyjściu standardowym, wyświetli największy czynnik rozkładu na czynniki pierwsze liczby n.

Ćwiczenie 3

Napisz program rozkładający liczbę naturalną dodatnią n na czynniki pierwsze, następnie wyświetlający na ekranie czynnik, który w jej rozkładzie wystąpił największą liczbę razy. Jeśli jest kilka takich czynników, niech program wypisze najmniejszy z nich. Sprawdź jego działanie dla `n = 13835745`.

Jeśli sprawdzana liczba jest liczbą pierwszą, program powinien wyświetlić komunikat `n to liczba pierwsza`.

Specyfikacja problemu:

Dane:

- n – liczba naturalna dodatnia; `n > 1`

Wynik:

Program na wyjściu standardowym wyświetla czynnik, który pojawił się najczęściej w rozkładzie na czynniki pierwsze liczby n, lub komunikat `n to liczba pierwsza`.

Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Faktoryzacja w języku Python

Grupa docelowa:

Liceum ogólnokształcące i technikum, 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:

2) wykorzystuje znane sobie algorytmy przy rozwiązywaniu i programowaniu rozwiązań następujących problemów:

a) rozkładania liczby na czynniki pierwsze,

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 rozkładu liczby na czynniki pierwsze.
- Zaimplementujesz algorytm rozkładu liczby na czynniki pierwsze.
- Przetestujesz program dla różnych 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 „Faktoryzacja w języku Python”.
2. Uczniowie przypominają sobie najważniejsze informacje dotyczące faktoryzacji.

Faza wstępna:

1. Chętna lub wybrana osoba przedstawia najważniejsze informacje związane z faktoryzacją liczb.
2. Nauczyciel wyświetla temat i cele zajęć. Prosi uczniów, by na podstawie wiadomości zdobytych przed lekcją zaproponowali kryteria sukcesu.

Faza realizacyjna:

1. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Film samouczek”. Uczniowie w parach wykonują polecenie 1. Następnie porównują swoje rozwiązanie z przedstawionym w filmie. Nauczyciel w razie potrzeby wyjaśnia niezrozumiałe kwestie.
2. **Praca z tekstem.** Nauczyciel wyświetla zawartość sekcji „Przeczytaj”. Uczniowie w grupach analizują przedstawioną w niej implementacje algorytmu rozkładu liczby na czynniki pierwsze w języku Python. Porównują czasy rozkładu dla danych liczb dla obu sposobów. Nauczyciel inicjuje dyskusję dotyczącą tego, co wpływa na czas wykonywania obliczeń.
3. **Ćwiczenie umiejętności.** Nauczyciel dzieli klasę na grupy czteroosobowe. Uczniowie wykonują ćwiczenie 1 z sekcji „Sprawdź się”. Następnie uczniowie na forum klasy omawiają rozwiązania.
4. Uczniowie indywidualnie wykonują ćwiczenie 2 z sekcji „Sprawdź się”. Następnie w parach porównują swoje rozwiązania.

Faza podsumowująca:

1. Na koniec zajęć nauczyciel raz jeszcze wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie”. W odniesieniu do ich realizacji dokonuje szczegółowej oceny rozwiązania zastosowanego przez wybranego ucznia.
2. Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności, omawia ewentualne problemy podczas rozwiązania ćwiczeń z programowania w języku Python.

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:

- Nauczyciel może wykorzystać multimedium w sekcji „Film samouczek” do pracy przed lekcją. Uczniowie zapoznają się z jego treścią i przygotowują do pracy na zajęciach w ten sposób, żeby móc samodzielnie rozwiązać zadania dołączone do e-materiału „Faktoryzacja w języku Python”.