Algorytmy tekstowe w języku Python

Źródło: Jessica Lewis, domena publiczna.

Algorytmy tekstowe są przydatne do przetwarzania oraz przeszukiwania danych tekstowych. Jeden z nich już poznaliśmy, to algorytm Knutha-Morrisa-Pratta służący do wyszukiwania wzorca w tekście.

Algorytmów tekstowych używamy, korzystając z edytora tekstu czy klienta e-mail. Znajdują również zastosowanie w grach. Możemy je wykorzystać, projektując implementację popularnej gry w wisielca. W e-materiale Algorytmy tekstowe omówiliśmy jej zasady oraz opracowaliśmy odpowiedni pseudokod.

W tym e-materiale zajmiemy się implementacją tej gry w języku Python.

Implementację algorytmów tekstowych w pozostałych językach programowania znajdziesz w e-materiałach:

- Algorytmy tekstowe w języku C++,
- Algorytmy tekstowe w języku Java.

> **Twoje cele**
>
> - Zaimplementujesz algorytmy tekstowe w języku Python.
> - Przećwiczysz operacje na tekstach.
> - Połączysz wiedzę o algorytmach z praktyką programistyczną.

Film samouczek

Problem 1

Napisz program symulujący grę w wisielca.

Specyfikacja problemu:

Założenia:

- program nie rysuje szubienicy – liczbę szans przechowuje w zmiennej;
- komputer losuje słowo z wcześniej przygotowanej tablicy;
- gracz zgaduje słowo poprzez wprowadzanie kolejnych liter; jeżeli litera znajduje się w słowie, powinien pojawić się komunikat informujący, że dana litera wystąpiła w słowie; jeśli nie, pojawia się stosowny komunikat i informacja na temat tego, ile szans pozostało graczowi;
- gra kończy się, gdy gracz zgadnie słowo w całości lub zabraknie mu szans.

Dane:

- `slowa` – tablica łańcuchów znaków

Wynik:

Program wyświetla komunikat na temat wygranej (`Gratulacje! Wygrałeś`), jeśli użytkownik odgadnie słowo, lub porażki (`Próbuj dalej. Przegrałeś`) w przeciwnym wypadku.

Uwaga, w programie wielkość liter ma znaczenie.

```
1
```

```
1
```

Polecenie 1

Porównaj swoje rozwiązanie z filmem.

Film dostępny pod adresem /preview/resource/RqLdrgwlEDfG0

Film nawiązujący do treści materiału

Kod programu zaprezentowanego w filmie:

Plik o rozmiarze 1.58 KB w języku polskim

Przeczytaj

Implementacja w języku Python – podsumowanie

Wiemy, jak przebiega partia gry w wisielca, spróbujmy więc napisać program, który ją odzwierciedli. Przedstawimy alternatywę dla realizacji kodu gry zaprezentowanej w filmie samouczku.

Pierwszym krokiem jest zadeklarowanie zmiennych:

- `zycia` - przechowuje pozostałą liczbę prób (pozostałe życia), służącą do odgadnięcia litery lub całego słowa, jej wartość będzie równa liczbie znaków w słowie do odgadnięcia,
- `slowo` - przechowuje słowo do odgadnięcia,
- `odgadniete` – przechowuje licznik odgadniętych znaków (liczbę punktów – za każdą odgadniętą literę przyznajemy 1 pkt.).

Kolejnym krokiem jest pobranie od pierwszego gracza słowa, które gracz drugi będzie próbował odgadnąć. Ciąg znaków pobrany z klawiatury zapisujemy w zmiennej `slowo`.

```
slowo = str(input("Podaj słowo do odgadnięcia: ")
```

Ustalamy liczbę żyć na tyle, ile wynosi liczba liter hasła do odgadnięcia. Ustawiamy również licznik odgadniętych znaków hasła na zero.

```
slowo = str(input("Podaj słowo do odgadnięcia: "))
zycia = len(slowo)
odgadniete = 0
```

Teraz należy zadeklarować pętlę, w której będzie się odbywała najważniejsza część algorytmu – pobieranie od drugiego gracza kolejnych liter (lub całego słowa). Program następnie sprawdzi, czy pobrane łańcuchy znaków są częścią podanego słowa.

```
while zycia > 0:
    odgadywane = str(input("Podaj literę lub całe słowo do sprawd
```

Ponieważ gracz może zgadywać pojedynczą literę lub całe słowo, to należy obsłużyć oba przypadki za pomocą instrukcji warunkowej `if ... else`.

```
if len(odgadywane) == 1:

else:
```

Zajmijmy się teraz pierwszym przypadkiem. Użytkownik odgaduje w nim pojedynczą literę. Wewnątrz instrukcji warunkowej sprawdzane jest, czy podana przez gracza litera występuje w wymyślonym słowie. Jeżeli występuje, zostaje wyświetlony odpowiedni komunikat, a zmienna zostaje inkrementowana o tyle, ile razy dana litera występuje w słowie.

Następnie program sprawdza, czy odgadnięte zostały wszystkie litery słowa.

W przeciwnym przypadku program również wyświetla odpowiedni komunikat, ale dekrementuje zmienną `zycia`.

```
if len(odgadywane) == 1:
    if odgadywane in slowo:
        print("Słowo zawiera literę:  ", odgadywane)
        odgadniete += slowo.count(odgadywane)
        if odgadniete == len(slowo):
            break
    else:
        print("Słowo nie zawiera litery:  ", odgadywane)
        zycia -= 1
```

W tym momencie użytkownik zdobywa punkt za każdym razem, gdy poda literę znajdującą się w słowie. Również wtedy, kiedy dana litera została już podana. Jak rozwiązać ten problem?

Odgadnięte litery będziemy dodawać do tablicy. Program będzie sprawdzał, czy zgadnięta litera została już podana. Jeśli tak, nie będzie przyznawał za nią punktu.

Najpierw zadeklarujemy zmienną `zgadniete_litery`, która będzie przechowywać zgadnięte litery. Zaimplementujmy pustą listę:

```
zgadniete_litery = []
```

Dodajmy odpowiedni warunek:

```
if len(odgadywane) == 1:
```

```
    if odgadywane in slowo:
        if odgadywane not in zgadniete_litery:
            print("Słowo zawiera literę:  ", odgadywane)
            odgadniete += slowo.count(odgadywane)
            zgadniete_litery.append(odgadywane)
        if odgadniete == len(slowo):
            break
    else:
        print("Słowo nie zawiera litery:  ", odgadywane)
        zycia -= 1
```

Teraz zdefiniujmy operacje, których wykonywanie nastąpi, gdy użytkownik nie wprowadzi pojedynczej litery tylko ciąg znaków.

Sprawdzamy, czy ciąg znaków podany przez drugiego gracza jest taki sam jak ciąg znaków wymyślony przez gracza pierwszego, tj. czy użytkownik poprawnie odgadł całe słowo. W przypadku, gdy wyrazy te są identyczne, oznacza to, że gracz drugi poprawnie odgadł słowo i następuje wyjście z pętli. W przeciwnym wypadku dekrementujemy zmienną `zycia`, która określa liczbę pozostałych prób.

```
if len(odgadywane) == 1:
    if odgadywane in slowo:
        if odgadywane not in zgadniete_litery:
            print("Słowo zawiera literę:  ", odgadywane)
            odgadniete += slowo.count(odgadywane)
            zgadniete_litery.append(odgadywane)
        if odgadniete == len(slowo):
            break
    else:
        print("Słowo nie zawiera litery:  ", odgadywane)
        zycia -= 1

else:
    if odgadywane == slowo:
        print("Podano poprawne słowo!")
        odgadniete = len(slowo)
        break
    else:
        print("Podane słowo nie jest hasłem")
        zycia -= 1
```

Po każdej iteracji pętli wyświetlana jest informacja o pozostałych próbach.

```
print("Pozostałe życia: ", zycia)
```

Ostatnie, co należy zrobić (już poza główną pętlą), to wyświetlić komunikat o przebiegu gry:

```
if odgadniete == len(slowo):
    print("Udało ci się odgadnąć słowo: ",slowo)
else:
    print("Niestety nie udało ci się odgadnąć słowa")
```

Pełny kod programu prezentuje się następująco:

```
slowo = str(input("Podaj słowo do odgadnięcia: "))
zycia = len(slowo)
odgadniete = 0
zgadniete_litery = []

while zycia > 0:
    odgadywane = str(input("Podaj literę lub całe słowo do sprawd

    if len(odgadywane) == 1:
        if odgadywane in slowo:
            if odgadywane not in zgadniete_litery:
                print("Słowo zawiera literę:  ", odgadywane)
                odgadniete += slowo.count(odgadywane)
                zgadniete_litery.append(odgadywane)
            if odgadniete == len(slowo):
                break
        else:
            print("Słowo nie zawiera litery:  ", odgadywane)
            zycia -= 1
    else:
        if odgadywane == slowo:
            print("Podano poprawne słowo!")
            odgadniete = len(slowo)
            break
        else:
            print("Podane słowo nie jest hasłem")
```

```
            zycia -= 1

    print("Pozostałe życia: ", zycia)

if odgadniete == len(slowo):
    print("Udało ci się odgadnąć słowo: ",slowo)
else:
    print("Niestety nie udało ci się odgadnąć słowa: ",slowo)
```

Polecenie 1

Zastanów się, o jakie funkcjonalności można rozszerzyć działanie programu.

Polecenie 2

Możemy wprowadzić element nieprzewidywalności i stworzyć grę, w której wyrazy do odgadnięcia będą dobierane losowo. Spróbujmy zdefiniować dodatkową funkcję, która wybierze słowo, a następnie uruchomi funkcję `wisielec()` z nim jako argumentem.

Słownik

ASCII

(ang. *American Standard Code for Information Interchange*) pierwotnie siedmiobitowy system kodowania znaków (współcześnie rozszerzony do ośmiu bitów); w oryginalnej wersji kodom z zakresu 0–127 przyporządkowano 26 liter łacińskich, 10 cyfr (0...9) oraz dodatkowe znaki

dekrementacja

operacja zmniejszenia wartości zmiennej o jeden; szeroko wykorzystywana w programowaniu np. jako zmiana wartości iteratora w pętli

instrukcja warunkowa

element algorytmu pozwalający na sprawdzenie jednego lub kilku warunków, a następnie zdefiniowanie, jakie czynności mają być wykonane, jeśli dane warunki są spełnione lub niespełnione (służy do sterowania programem)

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Rozpatrzmy grę w łańcuch słów. Gracze naprzemiennie podają słowa. Słowo podane przez gracza musi rozpoczynać się literą, którą kończyło się poprzednie słowo. Jeśli pierwszy gracz poda słowo „lokomotywa", drugi może podać np. słowo „arbuz". Napisz program, który będzie symulował zachowanie jednego z graczy.

Algorytm dysponować będzie tablicą znanych słów, a po podaniu słowa powinien znaleźć w tablicy słowo zgodne z regułami gry i zwrócić je. Jeśli słownik nie zawiera odpowiedniego słowa, funkcja powinna zwrócić słowo PRZEGRANA.

Specyfikacja problemu:

Dane:

- `znane_slowa` – tablica łancuchów znaków

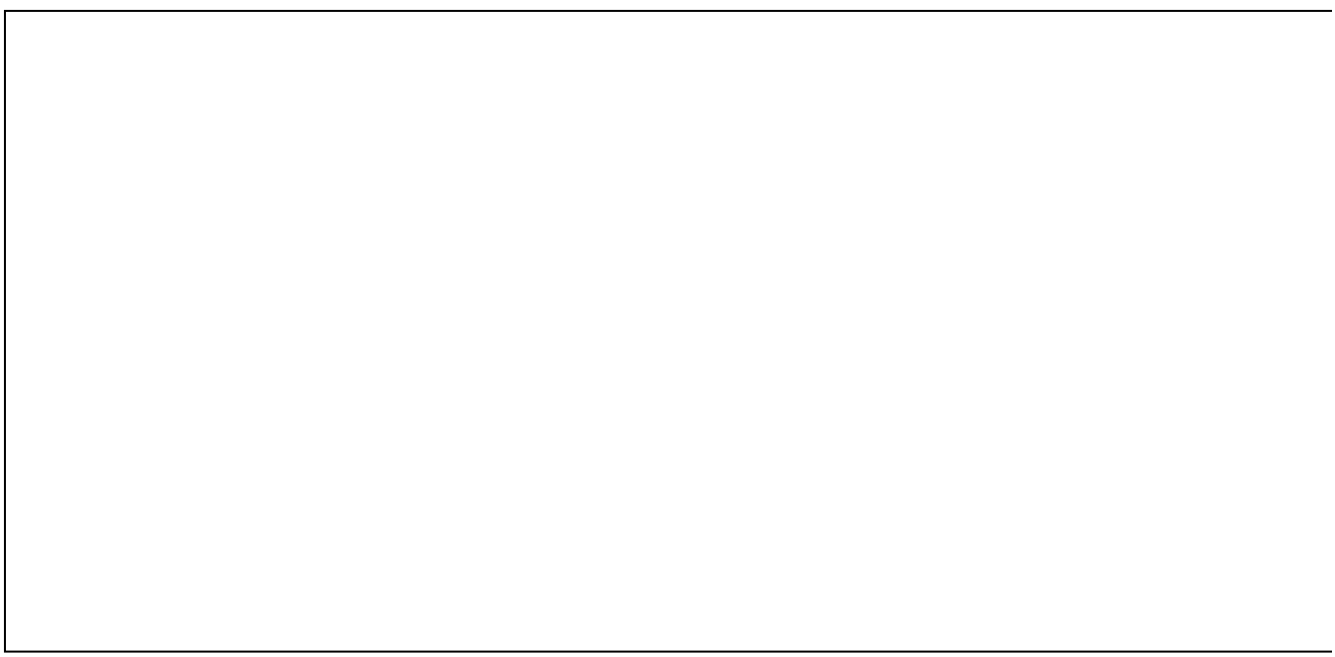
Wynik:

- komunikat PRZEGRANA lub odpowiednie słowo z tablicy

Uwaga, program rozpoznaje wielkość liter.

Twoje zadania

1. Program drukuje na standardowe wyjście wynik działania funkcji `lancuch_slow` dla wartości `wiadro`, `rebus`.



Ćwiczenie 2

Napisz program sprawdzający dla gry, w której uczestnicy układają słowa z liter tworzących inne słowo. Funkcja sprawdzająca ma przyjmować dwa słowa. Jeśli pierwsze z nich składa się tylko i wyłącznie z liter tworzących drugie słowo, funkcja powinna zwracać `True`, zaś w przeciwnym wypadku `False`.

Szczególne przypadki:

W słowie bazowym dana litera pojawia się x razy, natomiast w utworzonym słowie więcej niż x razy, np. litera ***a*** w słowie bazowym *skarb* i utworzonym *barak:*

```
False
```

W słowie bazowym dana litera pojawia się x razy, natomiast w utworzonym słowie mniej niż x razy, np. litera ***a*** w słowie bazowym *barak* i utworzonym *karb*:

```
True
```

Specyfikacja problemu:

Dane:

- `slowo, slowoBazowe` – łancuchy znaków

Wynik:

Program drukuje na standardowe wyjście wynik funkcji `sprawdz` (`True` lub `False`) dla przykładowych słów.

Uwaga, program rozpoznaje wielkość liter.

Twoje zadania

1. Program sprawdza, czy dane słowo zostało utworzone wyłącznie z liter podanego słowa bazowego.

Ćwiczenie 3

Przeprowadzana jest gra w wisielca – gracz próbuje odgadnąć słowo, którego litery są niewidoczne. Aby tego dokonać, może wybrać literę, która zostanie odsłonięta. Jeśli litera nie występuje w danym słowie, gracz traci jeden punkt życia. Algorytm symulujący przebieg gry skonstruowany jest według następujących zasad:

1. Pierwsza sprawdzana litera to „A” (pozycja w alfabecie 0, zatem `n = 0`).
2. Jeżeli sprawdzana litera, której pozycja w alfabecie to `n`, występuje w słowie `k` razy, następną sprawdzaną literą będzie litera na pozycji `n * 2 + k`.
3. Jeśli dana litera nie występuje w słowie, sprawdzana jest litera na pozycji `n + 1`.
4. Jeżeli pozycja kolejnej litery przekracza zakres alfabetu („Z” występuje na pozycji 25), wykonujemy dzielenie modulo 26 aktualnej pozycji).
5. W przypadku natrafienia na już sprawdzoną literę, nie sprawdzamy jej ponownie, a następną sprawdzaną literą będzie litera na pozycji `n + 1`.

Sprawdź, ile podanych w tablicy słów algorytm będzie w stanie odgadnąć. Zakładamy, że do dyspozycji jest `x` punktów życia.

Specyfikacja problemu:

Dane:

- `slowa` – tablica łańcuchów znaków
- `x` – liczba naturalna; liczba punktów życia

Wynik:

- `liczba` – liczba naturalna; liczba wygranych

Uwaga, program powinien rozpoznawać wielkość liter i w razie potrzeby dokonywać konwersji zapisu z małych liter na wielkie litery (lub na odwrót).

Twoje zadania

1. Program wypisuje liczbę naturalną `liczba` przechowującą informację na temat tego, ile słów jest w stanie odgadnąć algorytm.

Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Algorytmy tekstowe w języku Python

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres podstawowy i 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 podstawowy. Uczeń:

2) stosuje przy rozwiązywaniu problemów z różnych dziedzin algorytmy poznane w szkole podstawowej oraz algorytmy:

b) na tekstach: porównywania tekstów, wyszukiwania wzorca w tekście metodą naiwną, szyfrowania tekstu metodą Cezara i przestawieniową,

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

2) do realizacji rozwiązania problemu dobiera odpowiednią metodę lub technikę algorytmiczną i struktury danych;

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

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

2) stosuje zasady programowania strukturalnego i obiektowego w rozwiązywaniu problemów;

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

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):

- Zaimplementujesz algorytmy tekstowe w języku Python.
- Przećwiczysz operacje na tekstach.
- Połączysz wiedzę o algorytmach z praktyką programistyczną.

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: „Algorytmy tekstowe w języku Python". Uczniowie zapoznają się z treściami w sekcji „Film samouczek".

Faza wstępna:

1. Nauczyciel wyświetla uczniom temat, wskazuje cele zajęć oraz ustala z uczestnikami zajęć kryteria sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Uczniowie tworzą pytania dotyczące tematu zajęć, na które odpowiedzą w trakcie lekcji.

Faza realizacyjna:

1. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Film samouczek". Na forum klasy uczniowie analizują przedstawioną w niej implementację gry w wisielca w języku Python
2. **Praca z tekstem.** Nauczyciel wyświetla zawartość sekcji „Przeczytaj". Uczniowie zapoznają się z tekstem. Uczniowie implementują i testują omówiony program na swoich komputerach. Nauczyciel odpowiada na ewentualne pytania.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenie nr 1, 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. Na koniec zajęć z programowania w Pythonie nauczyciel prosi uczniów o rozwinięcie zdania: „Na dzisiejszych zajęciach nauczyłam/łem się jak…".

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:

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