Algorytm Knutha-Morrisa-Pratta w języku Python

Źródło: Wallace Chuck, domena publiczna.

W e-materiale Algorytm Knutha-Morrisa-Pratta przeanalizowaliśmy algorytm wyszukiwania wzorca w tekście. Zdobyte w ten sposób informacje teoretyczne spróbujemy teraz zastosować w praktyce. Dlatego w tym e-materiale zapoznamy się z implementacją algorytmu KMP w języku Python.

Implementację tego algorytmu w pozostałych językach programowania znajdziesz w e-materiałach:

- Algorytm Knutha-Morrisa-Pratta w języku C++,
- Algorytm Knutha-Morrisa-Pratta w języku Java.

Więcej zadań? Przejdź do e-materiału Algorytm Knutha-Morrisa-Pratta – zadania maturalne.

> **Twoje cele**
>
> - Przeanalizujesz metodę tworzenia tablicy częściowych dopasowań dla algorytmu KMP.
> - Zaimplementujesz funkcję realizującą wyszukiwanie wzorca w tekście za pomocą algorytmu KMP.
> - Rozwiążesz przykładowe zadania programistyczne z wykorzystaniem algorytmu KMP.

Przeczytaj

Algorytm KMP – implementacja w języku Python

Przykład 1

W materiale omawiającym algorytm Knutha-Morrisa-Pratta możemy znaleźć zapisany za pomocą pseudokodu algorytm oraz dokładną analizę jego działania. W tym e-materiale stworzymy implementację tego algorytmu. Najpierw napiszemy funkcję, której zadaniem będzie zbudowanie tablicy częściowych dopasowań. Tablicy tej potrzebujemy, aby efektywnie przeprowadzać próby dopasowań wzorca w tekście:

```
def budowa_tablicy_KMP(wzorzec):
    rozmiar_tablicy = len(wzorzec) + 1
    tablica = [0] * rozmiar_tablicy                          # d
    poz = tablica[0] = -1                                    # n

    for i in range(1, rozmiar_tablicy):                      # i

        while poz > -1 and wzorzec[poz] != wzorzec[i - 1]:   # d
            poz = tablica[poz]
        poz += 1

        if i == rozmiar_tablicy - 1 or wzorzec[i] != wzorzec[po
            tablica[i] = poz                                 # z
                                                             # k
                                                             # j
        else:
            tablica[i] = tablica[poz]                        # z
                                                             # n
    return tablica
```

Następnie zdefiniujmy funkcję, która, używając tablicy, będzie realizowała wyszukiwanie wzorca w tekście. Korzysta ona z wyżej zdefiniowanej funkcji do budowania tablicy częściowych dopasowań. W kodzie użyjemy funkcji `enumerate` w celu iterowania po znakach w tekście:

```
def szukaj_algorytmem_KMP(tekst, wzorzec):
```

```
    tablica_T = budowa_tablicy_KMP(wzorzec)
    dop = 0
    wynikowy_indeks = -1
    for indeks, znak_S in enumerate(tekst):
        while dop > -1 and wzorzec[dop] != znak_S:
            dop = tablica_T[dop]
        dop += 1

        if dop == len(wzorzec):
            wynikowy_indeks = indeks - len(wzorzec) + 1
            break
    return wynikowy_indeks
```

Pełna wersja programu wraz z kilkoma przykładami wykonań funkcji:

```
def budowa_tablicy_KMP(wzorzec):
    rozmiar_tablicy = len(wzorzec) + 1
    tablica = [0] * rozmiar_tablicy
    poz = tablica[0] = -1

    for i in range(1, rozmiar_tablicy):
        while poz > -1 and wzorzec[poz] != wzorzec[i - 1]:
            poz = tablica[poz]
        poz += 1

        if i == rozmiar_tablicy - 1 or wzorzec[i] != wzorzec[po
            tablica[i] = poz
        else:
            tablica[i] = tablica[poz]

    return tablica


def szukaj_algorytmem_KMP(tekst, wzorzec):

    tablica_T = budowa_tablicy_KMP(wzorzec)
    dop = 0
    wynikowy_indeks = -1
    for indeks, znak_S in enumerate(tekst):
        while dop > -1 and wzorzec[dop] != znak_S:
```

```
            dop = tablica_T[dop]
        dop += 1
        if dop == len(wzorzec):
            wynikowy_indeks = indeks - len(wzorzec) + 1
            break
    return wynikowy_indeks


# Działanie funkcji budującej tablicę dla przykładowych wzorców
print(budowa_tablicy_KMP("ADCDADC"))
# [-1, 0, 0, 0, -1, 0, 0, 3]

print(budowa_tablicy_KMP("AAAAAAAAA"))
# [-1, -1, -1, -1, -1, -1, -1, -1, -1, 8]

print(budowa_tablicy_KMP("ASDGORKASDF"))
# [-1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 3, 0]

print(budowa_tablicy_KMP("ASDFORKASDFO"))
# [-1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 5]

# Szukanie dopasowań wzorca w tekście
szukany_W = "ADCDADC"
lancuch_S = "ADC ADCDAD ADCDADCDADCE"
print(szukaj_algorytmem_KMP(lancuch_S, szukany_W))
# 11

# przykład sprawdzenia za pomocą wbudowanej metody find
print(lancuch_S.find(szukany_W))
# 11
```

Dla zainteresowanych

Metoda `find()` w języku Python realizuje wyszukiwanie tekstu za pomocą zmodyfikowanego algorytmu Boyer-Moor-Horspool – więcej informacji na ten temat można znaleźć w kodzie źródłowym CPython.

Słownik

enumerate()

funkcja w języku Python służąca do generowania indeksu i wartości odpowiadającej temu indeksowi w liście

metoda naiwna

określana również jako *brute force*; sprawdza wszystkie możliwe dopasowania wzorca w tekście

algorytm Boyer-Moore-Horsepool

jest to ulepszenie algorytmu Boyera-Moore'a; podobnie jak algorytm KMP analizuje wzorzec w celu znalezienia maksymalnego skoku w analizie tekstu po nieudanym dopasowaniu; porównanie wzorca z fragmentem tekstu odbywa się od końca wzorca; średnia złożoność czasowa tego algorytmu to $O = \left(\frac{\text{długość wzorca}}{\text{alfabet}} \right)$ i w praktyce jest on znacznie szybszy od rozwiązania naiwnego

Symulacja interaktywna

Problem 1

W języku Python napisz program, który wyszuka w tekście pierwszy fragment o zadanej długości, złożony jedynie z cyfr. Użyj algorytmu KMP. Program powinien zwrócić indeks początku znalezionego wzorca w tekście. Działanie programu przetestuj dla przykładowych danych:

```
tekst = "ab3def45ab12345hijk"
dlugosc_wzorca = 5
```

Specyfikacja:

Dane:

- `tekst` – przeszukiwany tekst; łańcuch znaków
- `dlugosc_wzorca` – długość szukanego fragmentu w tekście; dodatnia liczba całkowita

Wynik:

- `wynikowy_indeks` – indeks wskazujący w danym tekście na pierwszy znak znalezionego wzorca; jeżeli wzorzec nie został znaleziony, indeks jest równy −1; liczba całkowita

Twoje zadania

1. Program zwraca indeks wskazujący na początek znalezionego wzorca lub -1, jeżeli nie został on znaleziony w tekście.

```
def szukaj_algorytmem_KMP(tekst, dlugosc_wzorca):
    wynikowy_indeks = -1

    # tutaj dodaj swój kod

    return wynikowy_indeks

tekst = "ab3def45ab12345hijk"
```

```
9  dlugosc_wzorca = 5
10
11 print(szukaj_algorytmem_KMP(tekst, dlugosc_wzorca))
```

```
1
```

Polecenie 1

Porównaj swoje rozwiązanie z prezentacją.

1

Program będzie modyfikacją implementacji algorytmu KMP, którą przedstawiliśmy w sekcji „Przeczytaj”. W przypadku aktualnie rozwiązywanego problemu wzorzec posiada jednak ogólniejszą formę – jest ciągiem dowolnych cyfr. Zaczynamy od zdefiniowania funkcji, która przyjmuje dwa parametry:

przeszukiwany tekst – `tekst` oraz długość wzorca – `dlugosc_wzorca`. Zadeklarujemy również zmienną `wynikowy_indeks`, która będzie przechowywać zwracany indeks:

```
def szukaj_algorytmem_KMP(tekst, dlugosc_wzorca):
    wynikowy_indeks = -1

```

2

Następnie zajmiemy się kwestią tablicy częściowych dopasowań. Algorytm KMP używa tej tablicy, aby trzymać w niej informacje o budowie szukanego wzorca. Dzięki tablicy może określić, od którego indeksu wzorca powinien rozpocząć kolejną próbę dopasowania do tekstu, po wcześniejszym znalezieniu niedopasowania.

Rozpatrujemy wzorzec, który składa się z dowolnego ciągu cyfr o zadanej długości. Dlatego nie potrzebujemy go analizować. Jeżeli podczas próby dopasowania napotkamy na znak `c` niebędący cyfrą, to dowolna próba dopasowania, zawierająca `c`, będzie również niedopasowaniem. To znaczy, że możemy rozpocząć kolejną próbę dopasowania w tekście od znaku następującego po `c` i zaczniemy dopasowywanie wzorca od jego początkowego indeksu.

Takie działanie daje tablica częściowych dopasowań, która posiada na pierwszej pozycji wartownika `-1` oraz jest wypełniona w całości `0` – zawsze zaczniemy dopasowywanie od początku wzorca. Skoro jednak każda próba dopasowania będzie rozpoczynała się tak samo, możemy zastąpić tablicę pojedynczą liczbą, czyli `-1`. Nie musimy tego robić, jednak dla klarowności zadeklarujemy zmienną

`poczatek_wzorca`, która będzie spełniała tę rolę.

```
def
szukaj_algorytmem_KMP(teks
t, dlugosc_wzorca):
    wynikowy_indeks = -1
    poczatek_wzorca = -1

```

3

Kolejny fragment programu wygląda dokładnie tak samo, jak w implementacji algorytmu KMP. Deklarujemy zmienną `dop` wskazującą na kolejny element wzorca, dla którego ma zostać sprawdzone dopasowanie (więc początkowa wartość `0`). Tworzymy również pętlę `for`, w której będzie odbywała się iteracja po znakach w tekście.

```
def
szukaj_algorytmem_KMP(teks
t, dlugosc_wzorca):
    wynikowy_indeks = -1
    poczatek_wzorca = -1
    dop = 0
    for indeks, znak_S in
enumerate(tekst):
        pass

```

4

W następnym fragmencie kodu weryfikujemy, czy obecny `znak_S` pasuje do wzorca – w przypadku tego problemu sprawdzamy, czy jest cyfrą. Jeżeli nie jest, to znaleźliśmy niedopasowanie i cofamy się we wzorcu do miejsca, od którego zaczniemy kolejną próbę

dopasowania. Jak już ustaliliśmy, będzie to zawsze początek wzorca, więc `dop` przyjmie wartość `poczatek_wzorca`. W związku z tym możemy użyć instrukcji `if` zamiast pętli `while`, jak było to zrobione w przypadku implementacji algorytmu KMP.

```
1
2 def
  szukaj_algorytmem_KMP(teks
  t, dlugosc_wzorca):
3     wynikowy_indeks = -1
4     poczatek_wzorca = -1
5     dop = 0
6     for indeks, znak_S in
  enumerate(tekst):
7         if dop > -1 and
  not znak_S.isdigit():
8             dop =
  poczatek_wzorca
9
```

5

Następnie zwiększamy indeks `dop` o 1 i sprawdzamy, czy długość dopasowanej części wzorca jest równa długości całego wzorca. Jeżeli tak jest, to znaleźliśmy dopasowanie i możemy zapisać w zmiennej `wynikowy_indeks` pozycję, od której zaczyna się wystąpienie wzorca w tekście.

```
1 def
  szukaj_algorytmem_KMP(teks
  t, dlugosc_wzorca):
2     wynikowy_indeks = -1
3     poczatek_wzorca = -1
4     dop = 0
5     for indeks, znak_S in
  enumerate(tekst):
6         if dop > -1 and
  not znak_S.isdigit():
```

```
            dop =
poczatek_wzorca
        dop += 1
        if dop ==
dlugosc_wzorca:

wynikowy_indeks = indeks -
dlugosc_wzorca + 1

```

6

Na koniec implementujemy wyjście z pętli `for` w przypadku, kiedy znaleźliśmy dopasowanie oraz zwrócenie wartości `wynikowy_indeks`. Jeżeli wyjście z pętli nie nastąpiło przez instrukcję `break`, to dopasowanie nie zostało znalezione i `wynikowy_indeks` ma wartość `-1`.

```
def
szukaj_algorytmem_KMP(teks
t, dlugosc_wzorca):
    wynikowy_indeks = -1
    poczatek_wzorca = -1
    dop = 0
    for indeks, znak_S in
enumerate(tekst):
        if dop > -1 and
not znak_S.isdigit():
            dop =
poczatek_wzorca
        dop += 1
        if dop ==
dlugosc_wzorca:

wynikowy_indeks = indeks -
dlugosc_wzorca + 1
            break
    return wynikowy_indeks

```

Źródło: Contentplus.pl sp. z o.o., licencja: CC BY-SA 3.0.

Polecenie 2

Uruchom aplet realizujący wyszukiwanie wzorca w tekście. W aplecie znaki indeksowane są od 0 (czyli pierwszy znak ciągu ma indeks 0).

Źródło: Contentplus.pl sp. z o.o., licencja: CC BY-SA 3.0.

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Uzupełnij tekst brakującymi informacjami.

Algorytm KMP wymaga [ ] liczby niezbędnych porównań niż algorytm naiwny, więc zapewnia [ ] wydajność wyszukiwania pierwszego wystąpienia ciągu w tekście.
W tablicy częściowych dopasowań wartość pełniąca rolę wartownika to [ ].
W języku Python wbudowana funkcja wyszukująca indeks pierwszego wystąpienia wzorca w łańcuchu znaków to [ ].
Nazwiska autorów algorytmu KMP to kolejno: [ ], [ ], [ ].

Peter | mniejszej | 0 | Pratt | mniejszą | Kevin | większą | -1 | Maurice | in | Knuth | super | większej | szybki | 1 | Morris | find | search

Ćwiczenie 2

Uzupełnij tablicę częściowych dopasowań dla wzorca ABCABCDDEABC zgodnie z funkcją budowy tablicy dla algorytmu KMP:

Wartość	A	B	C	A	B	C	D	D	E	A	B	C
-1					0				-1			

Ćwiczenie 3

Napisz program, który zliczy w przekazanym tekście wszystkie wystąpienia wzorca. Wzorzec jest dowolnym ciągiem znaków o długości określonej przez zmienną `dlugosc_wzorca`, spełniającym następujące warunki:

- jeżeli `dlugosc_wzorca` jest parzysta, wzorzec składa się z dowolnych małych liter i dowolnych dwóch wielkich liter pośrodku, np. `werTTsug`,
- jeżeli `dlugosc_wzorca` jest nieparzysta, wzorzec składa się z dowolnych małych liter oraz jednej dowolnej wielkiej litery pośrodku, np. `reWsd`.

Do rozwiązania użyj implementacji algorytmu KMP, która została przedstawiona w sekcji „Przeczytaj” (przydatne może być również przeanalizowanie symulacji interaktywnej).

Przetestuj kod dla danych:

```
tekst = "72048agfds3423abcDDefg5902341asdfaweEEgafsdfawRRgdfsasc
dlugosc_wzorca = 8
```

Specyfikacja:

Dane:

- `tekst` – tekst dla poszukiwań wystąpień wzorca; łańcuch znaków
- `dlugosc_wzorca` – długość poszukiwanego wzorca; dodatnia liczba całkowita

Wynik:

- `wystapienia` – zliczone wystąpienia wzorca w tekście; nieujemna liczba całkowita

Twoje zadania

1. Program zlicza wszystkie wystąpienia wzorca w tekście.

```python
def szukaj_algorytmem_KMP(tekst, dlugosc_wzorca):
    wystapienia = 0

    # tutaj dodaj kod

    return wystapienia


tekst = "72048agfds3423abcDDefg5902341asdfaweEEgafsdfawRRgdfsasd"
dlugosc_wzorca = 8
print(szukaj_algorytmem_KMP(tekst, dlugosc_wzorca))
```

```
1
```

Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Algorytm Knutha-Morrisa-Pratta 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:

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

g) metodę haszowania (wyszukiwanie wzorca w tekście),

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 metodę tworzenia tablicy częściowych dopasowań dla algorytmu KMP.
- Zaimplementujesz funkcję realizującą wyszukiwanie wzorca w tekście za pomocą algorytmu KMP.
- Rozwiążesz przykładowe zadania programistyczne z wykorzystaniem algorytmu KMP.

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: „Algorytm Knutha-Morrisa-Pratta w języku Python". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj".

Faza wstępna:

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

Faza realizacyjna:

1. **Praca z tekstem.** Jeśli przygotowanie uczniów do lekcji jest niewystarczające, nauczyciel prosi uczniów o zapoznanie się z sekcją „Przeczytaj".
   Następnie uczniowie wykonują polecenie 1. Chętna lub wybrana osoba przedstawia swoje rozwiązanie i wyjaśnia otrzymany rezultat. Pozostali uczniowie weryfikują poprawność rozwiązania lub przedstawiają alternatywne.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Symulacja interaktywna", czyta treść problemu 1. Uczniowie indywidualnie wypracowują swoje rozwiązania, a następnie porównują je z przedstawionym w prezentacji.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują ćwiczenia nr 1 i 2 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność wykonanych zadań, omawiając je wraz z uczniami.
4. Uczniowie wykonują w parach ćwiczenie 3 z sekcji „Sprawdź się". Po zakończeniu zadania porównują swoje rozwiązanie z wynikiem innej grupy.

Faza podsumowująca:

1. Nauczyciel inicjuje dyskusję na temat praktycznego wykorzystania algorytmu KMP.
2. 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.

Praca domowa:

1. Uczniowie wykonują polecenie 2 z sekcji „Symulacja interaktywna".

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 „Symulacja interaktywna" 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 „Algorytm Knutha-Morrisa-Pratta w języku Python”.