ALGORYTMY I STRUKTURY DANYCH

ĆWICZENIA DO PREZENTACJI

MODUŁ 10

Altkom Akademia S.A., materiały własne

1 ALGORYTMY

ĆWICZENIE 1.1:

Porównanie złożoności algorytmu liczb pierwszych

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętność:
  - analizy złożoności algorytmu
  - wpływu złożoności algorytmu na czas jego wykonania

CELE I ZADANIA:

- Utwórz funkcję weryfikującą, czy podana liczba jest pierwsza
- Utwórz funkcję, która zliczy liczby pierwsze w danym zakresie
- Zmierz czas wykonania tej funkcji i zobacz, jak się on zmienia wraz ze wzrostem danych wejściowych

ALGORYTM WYKONANIA:

- Utwórz funkcję o nazwie *is_prime_number* sprawdzającą, czy podana liczba jest liczbą pierwszą (tzn. czy dzieli się tylko przez 1 i samą siebie)
- Utwórz drugą funkcję o nazwie *count_prime_numbers*, która zliczy, ile jest liczb pierwszych w przedziale od 2 do podanej liczby
- W implementacji wykorzystaj poprzednią funkcję
- Analizując kod obu funkcji postaraj się określić, jaką złożoność ma użyty algorytm
- Dokonaj pomiaru czasu wykonania tej funkcji
- Zobacz w jakim stopniu wzrośnie ten czas, przy wzroście danych wejściowych
- Zastanów się, czy da radę tak zmienić kod funkcji, aby zmniejszyć złożoność algorytmu

2 ALGORYTMY WYSZUKIWANIA

ĆWICZENIE 2.1:

Implementacja wybranego algorytmu wyszukiwania

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia poznasz wybrane algorytmy wyszukiwania i sposób ich działania

CELE I ZADANIA:

- Wybierz jeden z przedstawionych na prezentacji algorytmów wyszukiwania
- Zaimplementuj go w Pythonie
- Utwórz przykładowe dane i sprawdź jego działanie (zarówno dla elementu znajdującego się w sekwencji, jak i dla elementu, którego nie ma)

ALGORYTM WYKONANIA:

- Wybierz jeden z przedstawionych na prezentacji algorytmów wyszukiwania
- Wzorując się na opisie działania oraz przedstawionym pseudokodzie zaimplementuj go w Pythonie
- Utwórz testowy zestaw danych
  - dane powinny być unikalne i wybrane losowo
  - wybierz losowo jedną z wartości znajdujących się w zestawie danych
  - wybierz także wartość, której w zestawie nie ma
- Sprawdź działanie implementacji algorytmu
- Powtórz test kilkukrotnie dla różnych zestawów danych wejściowych

3 ALGORYTMY SORTOWANIA

ĆWICZENIE 3.1:

Implementacja wybranego algorytmu sortowania

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia poznasz wybrane algorytmy sortowania i sposób ich działania

CELE I ZADANIA:

- Wybierz jeden z przedstawionych na prezentacji algorytmów sortowania
- Zaimplementuj go w Pythonie
- Utwórz przykładowe dane i sprawdź działanie algorytmu

ALGORYTM WYKONANIA:

- Wybierz jeden z przedstawionych na prezentacji algorytmów sortowania
- Wzorując się na opisie działania oraz przedstawionym pseudokodzie zaimplementuj go w Pythonie
- Utwórz testowy zestaw danych
  - dane powinny być unikalne i wybrane losowo
- Sprawdź działanie implementacji algorytmu – wypisz dane przed i po sortowaniu
- Powtórz test kilkukrotnie dla różnych zestawów danych wejściowych

4 REKURENCJA

ĆWICZENIE 4.1:

Drzewo katalogów

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętność tworzenia funkcji wykorzystujących rekurencję

CELE I ZADANIA:

- Napisz funkcję rekurencyjną umożliwiającą wyrysowanie zawartości podanego katalogu w formie drzewa (niezależnie od tego ile jest w nim zagnieżdżonych podkatalogów i plików)

ALGORYTM WYKONANIA:

- Zadanie polega na wyrysowaniu drzewa podanego katalogu
- Przykładowo fragment takiego drzewa mógłby wyglądać następująco:

```
├── .idea
│   ├── .gitignore
│   ├── inspectionProfiles
│   │   └── profiles_settings.xml
│   ├── misc.xml
│   ├── modules.xml
│   ├── PYTH_ALGORYTMY.iml
│   └── workspace.xml
├── ex01_01
│   ├── primes.py
│   └── __init__.py
├── ex01_przyklady
│   ├── comparisons.py
│   └── __init__.py
├── ex02_01
│   ├── binary_search.py
│   ├── data.py
│   ├── interpolation_search.py
│   ├── linear_search.py
│   ├── __init__.py
│   └── __pycache__
│       └── data.cpython-311.pyc
```

- Do wyrysowania gałęzi drzewa katalogów i plików potrzebne będą symbole graficzne o następujących kodach:
  - spacja – 32
  - pozioma kreska – 9472
  - pionowa kreska – 9474
  - ostatnia gałąź (przypomina literę L) – 9492
  - odgałęzienie (przechylona litera T) – 9500
- Wykorzystując te symbole utwórz 4-znakowe fragmenty gałęzi, jak na rysunku poniżej:
  ' '
  '│ '
  '├─ '
  '└─ '
- Do poruszania się po systemie plików zaimportuj klasę *Path* z modułu *pathlib*
- Utwórz funkcję o nazwie *tree* z dwoma parametrami
  - pierwszy, obowiązkowy, reprezentuje ścieżkę początkową dla której tworzymy drzewo (jest on typu *Path*)
  - drugi, opcjonalny reprezentuje początkowy tekst (przedrostek) umieszczany na początku każdej linii drzewa (domyślnie pusty łańcuch)
- Utwórz listę zawartości katalogu (użyj metody *iterdir*)
- Każdy element katalogu powiąż z odpowiednim symbolem graficznym (z gałęzią T, a ostatni – z gałęzią L)
- Wypisz na ekranie przedrostek, symbol graficzny i nazwę elementu (atrybut *name*)
- Sprawdź, czy element jest podkatalogiem (metoda *is_dir*)
- W takiej sytuacji należy rekurencyjnie wywołać funkcję *tree*, pamiętając o dodaniu dodatkowego wcięcia – przedrostek należy poszerzyć
  - o symbol pionowej kreski oraz 3 spacji, o ile bieżący katalog nie był ostatni
  - w przeciwnym razie – o 4 spacje
- Przetestuj działanie funkcji na wybranym podkatalogu katalogu domowego
- Przykładowe wywołanie może wyglądać następujaco:
  `tree(Path.home() / 'PycharmProjects' / 'PYTH_ALGORYTMY')`

5 STRUKTURY DANYCH

ĆWICZENIE 5.1:

Implementacja kolejki LIFO

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętność implemetacji kolejek LIFO w Pythonie

CELE I ZADANIA:

- Zaimplementuj strukturę stosu (kolejkę LIFO) wykorzystując klasę *deque* z modułu *collections* oraz alternatywnie klasę *list*
- Porównaj efektywność obu rozwiązań

ALGORYTM WYKONANIA:

- Utwórz klasę o nazwie *Stack* pełniącą rolę stosu (kolejki LIFO)
- W implementacji użyj klasy *deque* z modułu *collections*
- W klasie zdefiniuj metody:
  - *push* – metoda wstawia podany element do kolejki na szczyt stosu
  - *pop* – metoda zwraca i usuwa element ze szczytu stosu
  - *__str__* – metoda umożliwi konwersję instancji kolejki na tekst i proste podejrzenie zawartości stosu
- Upewnij się, czy rzeczywiście elementy są zdejmowane ze stosu w kolejności odwrotnej do ich wstawiania
- Zmierz czas potrzebny do wstawienie zadanej liczby elementów na stos oraz czas potrzebny na zdjęcie tej samej liczby elementów ze stosu
- Porównaj obie wartości

- Powtórz te same działania, tym razem wykorzystując klasę *list* do implementacji stosu
- Czy jest jakaś przewaga jednego rozwiązania nad drugim?

ĆWICZENIE 5.2:

Implementacja kolejki FIFO

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętność implemetacji kolejek FIFO w Pythonie

CELE I ZADANIA:

- Zaimplementuj strukturę kolejki FIFO wykorzystując klasę *deque* z modułu *collections* oraz alternatywnie klasę *list*
- Porównaj efektywność obu rozwiązań

ALGORYTM WYKONANIA:

- Utwórz klasę o nazwie *Queue* pełniącą rolę kolejki FIFO
- W implementacji użyj klasy *deque* z modułu *collections*
- W klasie zdefiniuj metody:
  - *enqueue* – metoda wstawia podany element do kolejki
  - *dequeue* – metoda zwraca i usuwa element element z kolejki, który w niej przebywa najdłużej (pierwszy dodany)
  - *__str__* – metoda umożliwi konwersję instancji kolejki na tekst i proste podejrzenie jej zawartości
- Upewnij się, czy rzeczywiście elementy są wyjmowane z kolejki w kolejności ich wstawiania
- Zmierz czas potrzebny do wstawienie zadanej liczby elementów do kolejki oraz czas potrzebny na wyjęcie tej samej liczby elementów z kolejki
- Porównaj obie wartości

- Powtórz te same działania, tym razem wykorzystując klasę *list* do implementacji kolejki FIFO
- Czy jest jakaś przewaga jednego rozwiązania nad drugim?