Lista 4: Proste algorytmy rekurencyjne

Witold Dyrka

Kwiecień 2021

Rozwiązania zadań, o ile nie wskazano inaczej, proszę przedstawić w formie kodu źródłowego w Pythonie (plik tekstowy z rozszerzeniem `.py`). Pliki z kodem źródłowym należy umieszczać w kontenerze odbiorczym bezpośrednio (pliki spakowane nie będą akceptowane).

Każdy plik z kodem źródłowym proszę opatrzyć komentarzem informującym o autorstwie. Powyższy zapis jest traktowany jako równoważny oświadczeniu, że kod został napisany samodzielnie. W przeciwnym przypadku komentarz powinien określać rodzaj i zakres udziału zewnętrznego oraz precyzyjnie wskazywać jego źródła. Umiejętne *oraz* dobrze udokumentowane korzystanie ze źródeł zewnętrznych nie obniża wartości samego rozwiązania, jednak nadmierne lub, co gorsze, bezrefleksyjne użycie materiałów zewnętrznych może negatywnie wpłynąć na proces nauki programowania.

Kod powinien być zredagowany zgodnie z zasadami przedstawionymi na wykładzie. Dobre praktyki dotyczące redagowania kodu w Pythonie opisuje dokument *PEP8 – Style Guide for Python Code*[1].

Ponadto dla każdej z funkcji implementujących algorytm napisz funkcję testującą, która na kilku przykładach zademonstruje poprawne działanie testowanej funkcji. Funkcja testująca powinna wyświetlać czytelne komunikaty. Do automatyzacji weryfikacji poprawności warto wykorzystać asercje.

Przesyłany do oceny kod źródłowy powinien być opatrzony komentarzami. Ogólne zasady tworzenia komentarzy dokumentacyjnych w Pythonie zawiera dokument *PEP257 – Docstring Conventions*[2]. Polecamy trzymać się stylu komentarzy dokumentacyjnych Google[3] albo NumPy[4]. Porównanie obu stylów znajdziemy w dokumentacji generatora dokumentacji Sphinx[5].

[1] https://www.python.org/dev/peps/pep-0008/

[2] https://www.python.org/dev/peps/pep-0257/

[3] https://google.github.io/styleguide/pyguide.html

[4] https://numpydoc.readthedocs.io/en/latest/format.html#docstring-standard

[5] https://www.sphinx-doc.org/en/master/usage/extensions/napoleon.html

Zad. 1

Zaimplementuj znajdowanie największego wspólnego dzielnika dwu liczb całkowitych przy pomocy algorytmu Euklidesa metodą rekursji jako funkcję w języku Python. Algorytm nie został przedstawiony na wykładzie, jednak jest bardzo dobrze udokumentowany w podręcznikach i zasobach edukacyjnych internetu (choćby Wikipedii). Przedstawiając rozwiązanie, należy wskazać źródło. Pamiętaj o funkcji testującej (co najmniej trzy przypadki) oraz komentarzach (w tym komentarzach dokumentujących).

Zad. 2

Wyszukiwanie binarne sprawdza, czy zadana wartość znajduje się w zadanym zakresie uporządkowanej (posortowanej) tablicy i zwraca jej (zadanej wartości) pozycję (indeks). Aby zredukować liczbę operacji, algorytm porównuje zadaną wartość z wartością w środku zakresu i sprawdza, czy zadana wartość jest jej równa, mniejsza, czy większa. W pierwszym przypadku zwracamy wynik, w pozostałych powtarzamy wyszukiwanie binarne na odpowiedniej połowie zakresu. Formalnie algorytm zapiszemy następująco:

0. Zadane są: uporządkowana tablica *tab* o długości N, zakres przeszukiwania określony przez *początek* i *koniec* oraz szukana wartość x.
1. Jeśli *początek* $>$ *koniec*, algorytm kończy się, (nie) zwracając *nic*, ponieważ nie znaleziono szukanej wartości.
2. Niech *środek* $:=$ (*początek* $+$ *koniec*)//2, gdzie // oznacza dzielenie całkowitoliczbowe.
3. Jeśli *tab*[*środek*] $==$ x, algorytm zwraca *środek* jako indeks wartości x i kończy się.
4. Jeśli *tab*[*środek*] $>$ x, algorytm podstawia *koniec* $:=$ *środek* $-$ 1 i wraca do kroku 1.
5. Jeśli *tab*[*środek*] $<$ x, algorytm podstawia *początek* $:=$ *środek* $+$ 1 i wraca do kroku 1.

Zadanie polega na *rekurencyjnej* implementacji algorytmu wyszukiwania binarnego w postaci funkcji w języku Python. Funkcja powinna przyjmować jako parametr tablicę uporządkowanych wartości, zakres przeszukiwania oraz szukaną wartość, a zwracać indeks szukanej wartości w tablicy albo *None*. Przedstawiając rozwiązanie, należy wskazać źródło.

Pamiętaj o funkcji testującej (co najmniej trzy przypadki) oraz komentarzach (w tym komentarzach dokumentujących).

Zad. 3

Zaimplementuj zliczanie wystąpień zadanego elementu w tablicy (Zad. 4 na Liście 1) metodą rekursji jako funkcję w języku Python. Pamiętaj o funkcji testującej (co najmniej trzy przypadki) oraz komentarzach (w tym komentarzach dokumentujących).

Zad. 4

Przedstawiony na wykładzie algorytm sortowania szybkiego jest nieoptymalny dla krótkich zakresów. Aby poprawić wydajność, krótkie zakresy można sortować np. przy pomocy algorytmu sortowania przez wstawianie lub sortowania bąbelkowego, które to algorytmy w przypadku optymistycznym osiągają liniową czasową złożoność obliczeniową. Przerób podany w skrypcie wykładu kod *quicksort*a tak, by w momencie, gdy zakres do posortowania jest krótki, zamiast kolejnego zejścia rekurencyjnego wywoływał funkcję sortowania zaimplementowaną w Zad. 1 na Liście 3. Pamiętaj o funkcji testującej (co najmniej trzy przypadki) oraz komentarzach (w tym komentarzach dokumentujących).

Następnie zweryfikuj korzyści płynące z powyższej optymalizacji. W tym celu utwórz losowe tablice elementów, np. używając funkcji `random.sample` z modułu `random`. Poniższy kod wylosuje listy odpowiednio 10, 100, 1 000 i 10 000 liczb naturalnych z zakresu od 1 do miliona, bez powtórzeń:

```
import random
lista_losowa_1 = random.sample(range(1, int(1e6)), int(1e1))
lista_losowa_2 = random.sample(range(1, int(1e6)), int(1e2))
lista_losowa_3 = random.sample(range(1, int(1e6)), int(1e3))
lista_losowa_4 = random.sample(range(1, int(1e6)), int(1e4))
```

Porównaj czasy realizacji sortowania dla różnych długości tablic z wykorzystaniem:

- algorytmu sortowania z Zad. 1 na Liście 3,
- czystego *quicksort*a z wykładu,
- połączenia obu powyższych.

Warto z funkcji implementujących powyższe algorytmy utworzyć moduł języka Python.

Pamiętaj, że algorytmy sortowania są zwykle implementowane jako algorytmy *in situ*, tzn. modyfikują podaną jako argument wejściowy tablicę. Dlatego kolejne algorytmy powinny operować na jej kopiach, co można osiągnąć na przykład tak, jak w poniższym fragmencie kodu:

```
sortowanie_szybkie(list(lista_losowa_1), 0, len(lista_losowa_1) - 1)
```

W pomiarze czasu przydatny będzie moduł `time`:

```
import time
start = time.time()
# Testowany kod
czas = time.time() - start
```

Dla każdego z dwunastu przypadków (trzy algorytmy, cztery długość tablicy) uśrednij czas dla sześciu wywołań.[6] Wykonaj sprawozdanie z eksperymentów obliczeniowych, które będzie zawierało opis badanego problemu, tabele z wynikami oraz komentarz do nich.

Zad. 5*

Dla zadań 1–3 napisz funkcje rekurencyjne tak, by nie korzystały z instrukcji podstawienia.

[6] Łączna liczba wywołań to $3 \cdot 4 \cdot 6 = 72$.