

Programowanie 1R – Ćwiczenia 2

Listy i krotki

Mikołaj Marcinkowski

3 marca 2026

Zadania możesz wykonywać jako oddzielne skrypty `.py` lub w notatniku jupytera `.ipynb`, chyba że treść zadania wskazuje inaczej.

Zadania wzorowane na Serii 2 zadań autorstwa Bartłomieja Zglinickiego (<https://www.fuw.edu.pl/~bzglinicki/teaching/plr/zadania/seria-2/>), udostępnionej na licencji CC BY 4.0.

Zadanie 1. Sortowanie bąbelkowe

Napisz funkcję, która przyjmuje listę (na przykład liczb) i zwraca listę jej elementów ułożonych w kolejności rosnącej wykorzystując algorytm sortowania bąbelkowego (ang. *bubble sort*).

Sortowanie bąbelkowe polega na przebieganiu raz za razem po elementach listy i porównywaniu sąsiednich elementów. Jeśli lewy element jest mniejszy od prawego, idziemy dalej. Jeśli prawy jest mniejszy, zamieniamy je miejscami i idziemy dalej.

Gdy w trakcie pełnego przejścia nie dojdzie do żadnych zamian, to znaczy, że lista jest już posortowana.

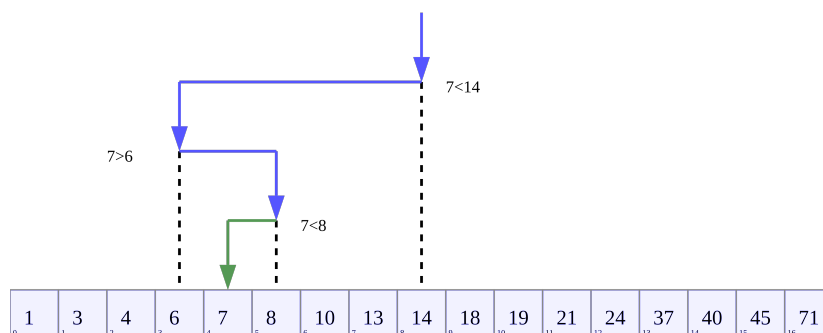
Przydatna informacja: W Pythonie operacje `<`, `>`, `<=` itp. oprócz porównywania liczb mogą też służyć do porównywania zmiennych tekstowych (stringów). Wtedy sprawdzana jest kolejność alfabetyczna.

Zadanie 2. Wyszukiwanie binarne

Wyszukiwanie binarne (ang. *binary search*) to szybki algorytm do sprawdzania, czy dany element znajduje się w posortowanej liście i jeśli tak, zwraca jego indeks. Opiera się on na następującej idei:

Porównujemy szukaną wartość z elementem w połowie listy. Jeśli jest mniejsza, to znaczy, że musi się znajdować po jego lewej stronie. Jeśli większa, to po prawej. W ten sposób jednym porównaniem zawęziliśmy obszar poszukiwań o połowę.

Z wybranej połówki znów bierzemy środkowy element i porównujemy z szukaną wartością, co znowu zawęży obszar o połowę. Tę procedurę powtarzamy, aż zawężymy obszar poszukiwań do jednego elementu. Jeśli to on, zwracamy jego indeks, jeśli nie, to szukana wartość nie zawierała się w liście.



Wyszukiwanie binarne w akcji. Autor rysunku AlwaysAngry CC BY 4.0

Napisz funkcję, która przyjmuje listę i szukaną wartość, po czym sortuje listę, szuka w niej tej wartości wyszukiwaniem binarnym i zwraca znaleziony indeks. Jeśli wartości nie było, powinna zwrócić -1 i może poinformować użytkownika o niepowodzeniu. Jeśli wartość pojawia się kilkakrotnie, funkcja powinna zwrócić indeks pierwszego wystąpienia.

Algorytm możesz zrealizować rekurencyjnie. Wtedy dodaj do funkcji dwa nowe argumenty, wskazujące indeksy początku i końca obszaru poszukiwań.

Zadanie 3. Najliczniejszy element

Napisz funkcję, która przyjmuje listę, sprawdza, który jej element występuje najliczniej, po czym zwraca krotkę (`najliczniejszy_element`, `liczba_wystąpień`). Jeśli kilka elementów zremisuje, niech zwróci listę takich krotek dla każdego z nich, w dowolnej kolejności (ale bez powtórzeń).

Wskazówka: Listy i krotki posiadają metodę `.count()`. Można też obejść się bez niej.

Zadanie 4. Wykres szybkości Fibonacciego

W zadaniu 8 z serii 1 porównywaliśmy czasy obliczenia n -tego wyrazu ciągu Fibonacciego algorytmem rekurencyjnym i iteracyjnym. Napisz program, który oblicza te czasy dla wartości n z wybranego przedziału, po czym przedstawia je na wykresie, którego oś x pokazuje numer wyrazu n , a oś y czasy obliczenia. Do zrobienia wykresu skorzystaj z biblioteki `matplotlib`.