Lista 4

Zadanie 1 (0,8 punktu) Zaproponuj i opisz taką modyfikację klasy `LinkedList` (oraz używanej przez niej klasy `Node`) która implementuje operację usuwania węzła z końca łańcucha w czasie $\Theta(1)$. Czasy pozostałych operacji dodawania i usuwania elementów powinny pozostać w czasie $\Theta(1)$.
Wskazówka: odpowiednia modyfikacja to tak zwana *lista podwójnie łączona*.

Zadanie 2* (0,2 punktu) Zaproponuj i opisz taką modyfikację klasy z poprzedniego zadania, która implementuje operację odwrócenia kolejności jej elementów w czasie $\Theta(1)$.

Zadanie 3 (1 punkt) Zaproponuj działanie (podaj szczegółowy opis) następujących operacji na liście łączonej (dla wersji z wykładu lub z poprzednich zadań):

- Sprawdzenie, czy podany obiekt znajduje się na liście łączonej (czyli jest pamiętany przez pewien węzeł).
- Wstawienie nowego węzła na n-tą pozycję od początku listy łączonej, gdzie n jest podane.
- Połączenie dwóch list łączonych w jedną.

Podaj złożoność czasową tych operacji.

Zadanie 4 (1 punkt) Zaimplementuj jako iteratory (przez klasy lub funkcje generujące):

- `fibonacci()`, zwracający kolejne elementy ciągu Fibonacciego $(0, 1, 1, 2, \ldots)$.
- `pascal(n)`, gdzie `n` to liczba naturalna, zwracający kolejne liczby z kolejnych wierszy trójkąta Pascala, do `n`-tego wiersza włącznie.
- `substrings(s)`, gdzie `s` to napis, zwracający wszystkie podnapisy `s` (napisy złożone z **kolejnych** znaków `s`) w dowolnej kolejności, każdy tylko raz[1]. Przykładowo, dla napisu `"abc"` iterator powinien zwracać napisy `""`, `"a"`, `"b"`, `"c"`, `"ab"`, `"bc"` i `"abc"`.
- (* nieobowiązkowe) `subsets(s)` (gdzie `s` to zbiór), zwracający wszystkie podzbiory `s` w dowolnej kolejności, każdy tylko raz.

Iteratory nie powinny konstruować (często długich) list (słowników, etc.) zawierających zwracane elementy.

Zadanie 5 (1 punkt) Uzupełnij klasę `LinkedList` tak, aby jej instancje były iterowalne:
(a) Przez napisanie iteratora jako klasy.
(b) Przez zaimplementowanie generatora (z użyciem `yield`).
Zaimplementuj też (w możliwie krótki sposób) metody `min()` i `max()` zwracające odpowiednio minimum i maksimum z elementów listy (przy założeniu, że elementy listy są parami porównywalne). Możesz założyć, że w czasie iteracji lista łączona nie ulega zmianom. Sama iteracja nie powinna modyfikować listy.
Wskazówka: zapoznaj się z odpowiednimi implementacjami iteracji dla klasy `Stack` w skrypcie do wykładu.

[1] W tym zadaniu przydatne może okazać się zapoznanie z instrukcją `yield from`, choć nie jest to konieczne.