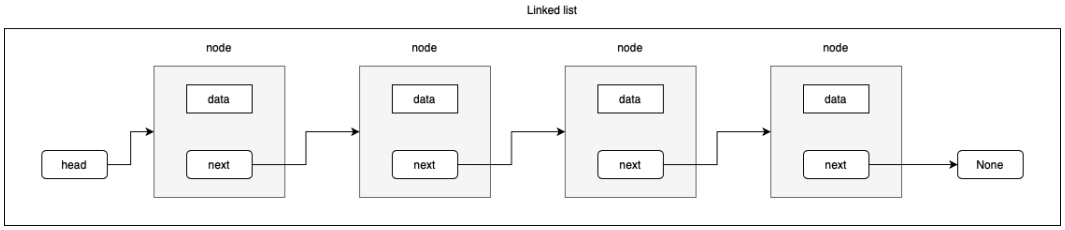
Algorytmy i struktury danych (studia stacjonarne)

Dr Anna Muranova

Semestr zimowy 2025/2026, UWM w Olsztynie

Ćwiczenie 5 (Lista jednokierunkowa) 12.11.2025

Zadanie 5.1. Lista jednokierunkowa jest sekwencją połączonych ze sobą węzłów. Każdy z nich zawiera wartość (`data` lub `value`) oraz wskaźnik do następnego elementu (`next`). Wskaźnikiem do pierwszego elementu listy jest `head`.



Przygotować implementację listy jednokierunkowej wg następujących wymagań:

(A) klasa `Node` będzie reprezentacją węzła z polami

```
class Node:

value: Any
next: Node
```

i następnymi metodami:

(a) funkcja `__str__` zwraca *value*− > jeżeli `next` nie jest **None**, oraz *value* jeżeli jest **None**

(b) metoda `insert_next(self, other) -> None` wstawia nowy węzeł tuz za węzłem, wskazanym w parametrze.

(c) metoda `remove_next(self) -> None` usuwa węzeł tuz za węzłem, wskazanym w parametrze.

(B) klasa `LinkedList` będzie reprezentacją listy jednokierunkowej z jednym polem `head: Node`.

(a) funkcja `len` wywołana na obiekcie listy zwróci jej długość

(b) funkcja `print` wywołana na obiekcie listy zawierającym 2 elementy $[0, 1]$ wyświetli na ekranie `"0 -> 1"`

(c) metoda `push(self, value: Any) -> None` umieści nowy węzeł na początku listy

(d) metoda `append(self, value: Any) -> None` umieści nowy węzeł na końcu listy

(e) metoda `node(self, at: int) -> Node` zwróci węzeł znajdujący się na wskazanej pozycji

(f) metoda `pop(self) -> Any` usunie pierwszy element z listy i go zwróci

(g) metoda `remove(self) -> Any` usunie ostatni element z listy i go zwróci

Proponowany schemat testów:

(a) Węzły:

```
node1 = Node(5)
assert str(node1) == '5'
node0 = Node('Ala',node1)
assert str(node0) == 'Ala -> '

node0.insert_next(Node(2))
assert node0.next.value == 2
node0.remove_next()
assert node0.next.value == 5
```

(b) Nowa lista jest pusta, długość równa się 0.

```
list_ = LinkedList()

assert list_.head == None
assert len(list_)==0
```

(c) Metoda `push` umieszcza elementy na początku listy.

```
list_.push(1)
list_.push(0)

assert list_.head.value == 0
assert str(list_) == "0 -> 1"
assert len(list_) == 2
```

(d) Metoda `append` umieszcza elementy na końcu listy

```
list_.append(9)
list_.append(10)

assert str(list_) == "0 -> 1 -> 9 -> 10"
assert len(list_) == 4
```

(e) Metoda `node` zwraca węzeł o podanym indeksie

```
assert str(list_.node(at=0)) == "0 -> "
assert str(list_.node(at=1)) == "1 -> "
assert list_.node(at=3).value == 10
assert list_.node(at=3).next is None
assert isinstance(list_.node(at=3), Node)
assert list_.node(at=4) is None
assert list_.node(at=10) is None
```

(f) Metoda `pop` usuwa i zwraca wartość pierwszego węzła listy.

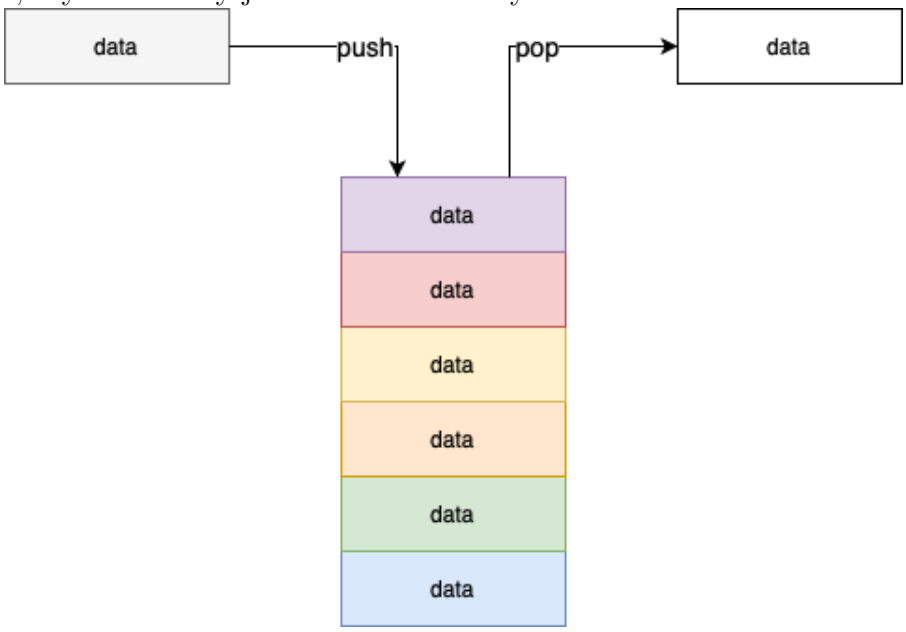
```
first_element = list_.node(at=0)
returned_first_element = list_.pop()
assert first_element.value == returned_first_element
assert str(list_) == "1 -> 9 -> 10"
```

(g) Metoda `remove` usuwa i zwraca ostatni element listy

```
last_element = list_.node(at=2)
returned_last_element = list_.remove()

assert last_element.value == returned_last_element
assert str(list_) == "1 -> 9"

list_.remove()

assert list_.remove() == 1
assert list_.head is None
assert len(list_) == 0
```

Zadanie 5.2. Stos - kolejka LIFO (last input, first output)

Elementy w stosie są układane jeden na drugim, co oznacza że nowa wartość znajdzie się na końcu kolejki. Przy usuwaniu źdejmowany"jest ostatni element kolejki, czyli ten który jest obecnie ńa szczycie".



Przygotować implementację stosu według następujących wymagań:

(a) klasa `Stack` będzie reprezentacją stosu do przechowywania wartości wewnątrz stosu wykorzystać własną implementację klasy `LinkedList`

```
class Stack:
_storage: LinkedList
```

(b) zaimplementuj metody `__len__` oraz `__str__`, która zwraca elementy w postaci kolumny

(c) wywołanie inicjalizatora klasy `Stack` utworzy pusty stos

```
stack = Stack()

assert len(stack) == 0
```

(d) metoda `push(self, element: Any) -> None` umieści nową wartość ńa szczycieństosu, czyli zostanie dodana na końcu wewnętrznej listy

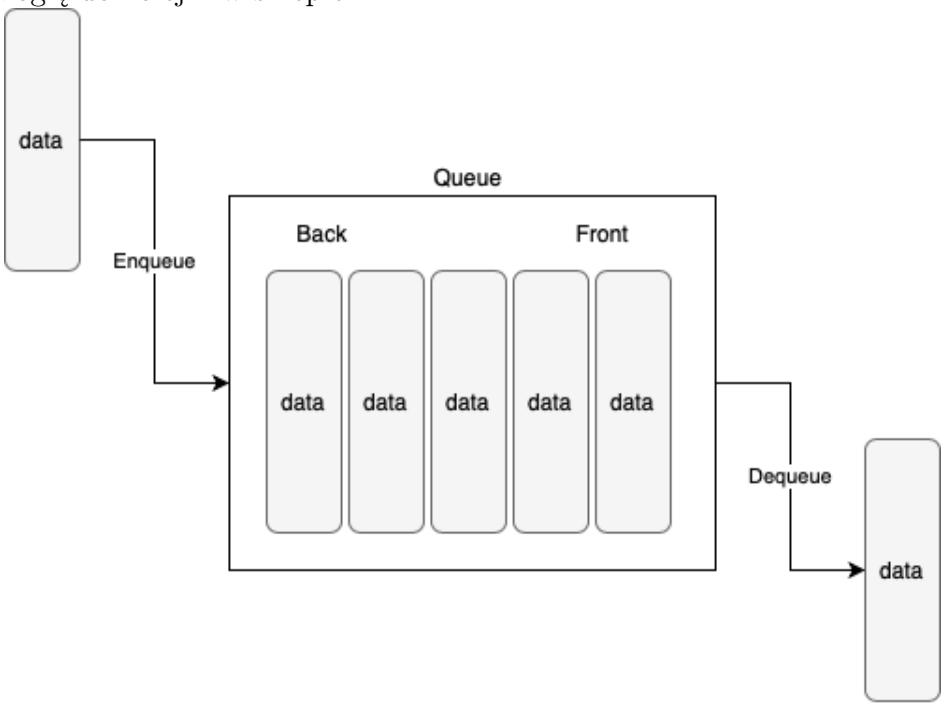
```
stack.push(3)
stack.push(10)
stack.push(1)

assert len(stack) == 3
assert str(stack)=="1\n10\n3\n"
```

(e) metoda `pop(self) -> Any` zwróci i usunie wartość ze szczytu stosu

```
top_value = stack.pop()

assert top_value == 1
assert len(stack) == 2

stack.pop()
stack.pop()

assert len(stack) == 0
```

Zadanie 5.3. Kolejka FIFO (first input, first output)

Kolejka FIFO to abstrakcyjna struktura danych pełniąca rolę bufora. Element dodany jako pierwszy zostaje również źdjęty"jako pierwszy, co można rozumieć przez analogię do kolejki w sklepie.



Przygotować implementację kolejki FIFO według następujących wymagań:

(a) klasa `Queue` będzie reprezentacją kolejki FIFO do przechowywania wartości wewnątrz kolejki FIFO wykorzystać własną implementację klasy `LinkedList`

```
class Queue:
_storage: LinkedList
```

(b) zaimplementuj metody `__len__` oraz `__str__`, która zwraca elementy w postaci wiersza

(c) wywołanie inicjalizatora klasy `Queue` utworzy pustą kolejką

```
queue = Queue()

assert len(queue) == 0
```

(d) metoda `enqueue(self, element: Any) -> None` umieści nowy element na końcu kolejki

```
queue.enqueue("klient1")
queue.enqueue("klient2")
queue.enqueue("klient3")

assert str(queue) == "klient1, klient2, klient3"
assert len(queue) == 3
```

(e) metoda `peek(self) -> Any` zwróci wartość pierwszego elementu w kolejce

```
assert queue.peek() == "klient1"
assert len(queue) == 3
```

(f) metoda `dequeue(self) -> Any` zwróci i usunie pierwszy element w kolejce

```
client_first = queue.dequeue()
assert client_first =="klient1"
assert str(queue) == "klient2, klient3"
assert len(queue) == 2

queue.dequeue()
queue.dequeue()
assert len(queue) == 0
```