Algorytmy i struktury danych laboratorium 3
Stos metody push i pop

Paweł Zabczyński

2020
Czerwiec

Definicje

Definicja elementu listy

```
class Node(Generic[T]):
    def __init__(self, value: T):
        self.next: Node[T] = None
        self.previous: Node[T] = None
        self.value: T = value
```

```
class List(Generic[T]):
    def __init__(self):
        self.head: Node[T] = None
        self.tail: Node[T] = None
        self.size: int = 0

    def is_empty(self):
        return self.head is None
```

```
class ComparableNode(Generic[T]):
    def __init__(self, value: T):
        self.next: ComparableNode[T] = None
        self.previous: ComparableNode[T] = None
        self.value: T = value

    def ==(self, other: ComparableNode[T]) -> bool:
        if other is None:
            return False
        return self.value == other.value

    def !=(self, other):
```

```
        if other is None:
            return True
        return self.value != other.value

    def >(self, other: ComparableNode[T]'):
        return self.value > other.value

    def >=(self, other: ComparableNode[T]):
        return self.value >= other.value

    def <(self, other: ComparableNode[T]):
        return self.value < other.value

    def <=(self, other: ComparableNode[T]):
        return self.value <= other.value
```

```
class SortedList(Generic[T]):
    def __init__(self):
        self.head: ComparableNode[T] = None
        self.tail: ComparableNode[T] = None
        self.size: int = 0

    def is_empty(self) -> bool:
        return self.head is None
```

Metody push

Metoda dodaje element na początek listy

Listing 1: push_front() lista jednokierunkowa

```
def push_front(xs, e: T):
    node = Node[T](e)
    xs.size += 1
    if xs.head is None:
        xs.head = node
        xs.tail = node
    else:
        node.next = xs.head
        xs.head = node
```

Listing 2: push_front() lista dwukierunkowa

```
def push_front(xs: List[T], e: T):
    node = Node[T](e)
```

```
    xs.size += 1
    if xs.head is None:
        xs.head = node
        xs.tail = node
    else:
        xs.head.previous = node
        node.next = xs.head
        xs.head = node
```

Listing 3: push_back() lista jednokierunkowa

```
def push_back(xs: List[T], e: T):
    node = Node[T](e)
    xs.size += 1
    if xs.head is None:
        xs.head = node
        xs.tail = node
    else:
        xs.tail.next = node
        xs.tail = node
```

Listing 4: push_back() lista dwukierunkowa

```
def push_back(xs: List[T], e: T):
    node = Node[T](e)
    xs.size += 1
    if xs.head is None:
        xs.head = node
        xs.tail = node
    else:
        node.previous = xs.tail
        xs.tail.next = node
        xs.tail = node
```

Metody pop

Metody pop - usuwające element list

Listing 5: pop_front() lista jednokierunkowa

```
def pop_front(xs: List[T]):
    node = None
    if xs.is_empty():
        raise IndexError
    elif xs.size == 1:
        xs.size -= 1
```

```
        node = xs.head
        xs.head = None
        xs.tail = None
    else:
        xs.size -= 1
        node = xs.head
        xs.head = xs.head.next

    return node.value
```

Listing 6: pop_front() lista dwukierunkowa

```
def pop_front(xs: List[T]):
    node = None
    if xs.is_empty():
        raise IndexError
    elif xs.size == 1:
        xs.size -= 1
        node = xs.head
        xs.head = None
        xs.tail = None
    else:
        xs.size -= 1
        node = xs.head
        xs.head = xs.head.next
        xs.head.previous = None

    return node.value
```

Listing 7: pop_back() lista jednokierunkowa

```
def pop_back(xs: List[T]):
    node_result = None
    if xs.is_empty():
        raise IndexError
    elif xs.size == 1:
        xs.size -= 1
        node_result = xs.head
        xs.head = None
        xs.tail = None
    else:
        node = xs.head
        while node.next != xs.tail:
            node = node.next

        node_result = node.next
```

```
        xs.tail = node
        xs.tail.next = None
        xs.size -= 1

        return node_result.value
```

Listing 8: pop_back() lista dwukierunkowa

```
def pop_back(xs: List[T]):
    node = None
    if xs.is_empty():
        raise IndexError
    elif xs.size == 1:
        node = xs.head
        xs.head = None
        xs.tail = None
    else:
        node = xs.tail
        xs.tail = node.previous
        xs.tail.next = None
    xs.size -= 1

    return node.value
```

Listing 9: pop_1() lista jednokierunkowa

```
    def pop_1()(xs: List[T], e: T):
        if xs.is_empty():
            raise IndexError

        result = None
        node = xs.head
        if xs.head.value == e:
            result = xs.head
            xs.head = xs.head.next
            xs.size -= 1
        else:
            while node.next and node.next.value != e:
                node = node.next

            if node.next.value == e:
                result = node.next
                xs.size -= 1
                if node.next.next is not None:
                    node.next = node.next.next
                else:
```

```
                xs.tail = node
                node.next = None
        else:
            result = None

    return result.value if result else None
```

Listing 10: pop_2() lista dwukierunkowa

```
def pop_2(xs: List[T], e: T):

    if xs.is_empty():
        raise IndexError

    result = None
    node = xs.head
    if node.value == e:
        result = xs.head
        xs.head = None
        xs.tail = None
        xs.size -= 1
    else:
        while node.next and node.next.value != e:
            node = node.next

        if node.next.value == e:
            result = node.next
            xs.size -= 1
            if node.next.next is not None:
                node.next = node.next.next
                node.next.previous = node
            else:
                xs.tail = node
                node.next = None
        else:
            result = None
```

Lista sortowane insert, delete

Listing 11: insert() lista jednokierunkowa

```
def insert(xs: SortedList[T], e: T):
    new_node = ComparableNode[T](e)
    xs.size += 1
    if xs.is_empty():
```

```
            xs.head = new_node
            xs.tail = new_node
        elif xs.head > new_node:
            new_node.next = xs.head
            xs.head = new_node
        else:
            node = self.head
            while node.next and node.next < new_node:
                node = node.next

            if node.next:
                new_node.next = node.next
                node.next = new_node
            else:
                node.next = new_node
                xs.tail = new_node
```

Listing 12: delete() lista jednokierunkowa

```
def delete(xs: SortedList[T], e: T) -> None:
        if xs.is_empty():
            raise IndexError

        result = None
        node = xs.head
        if xs.head.value == e:
            result = xs.head
            xs.head = xs.head.next
            xs.size -= 1
            if xs.size == 0:
                xs.tail = None
        else:
            while node.next and node.next.value != e:
                node = node.next

            if node.next.value == e:
                result = node.next
                xs.size -= 1
                if node.next.next is not None:
                    node.next = node.next.next
                else:
                    xs.tail = node
                    node.next = None
            else:
                result = None
```

Podsumowanie

Przykadowa implementacja list dwukierunkowej wraz z jednokierunkową GitHub