Programování 2

6. cvičení, 24-3-2022

tags: `Programovani 2`, `čtvrtek 1`, `čtvrtek 2`

Farní oznamy

1. Tento text a kódy ke cvičení najdete v repozitáři cvičení na https://github.com/PKvasnick/Programovani-2.
2. **Domácí úkoly**: Nové do konce týdne
    1. Hezky přibývají řešení

Ještě mám pořád skluz v kontrole.

Dnešní program:

- Kvíz
- Opakování: Lineární spojovaný seznam
- Varianty LSS: zásobník, fronta, cyklický zásobník, dvojitě spojovný seznam

---

Na zahřátí

> *"Dokonalost nevzniká tam, kde už není co přidat, ale tam, kde už nelze nic ubrat. " – Antoine de Saint-Exupery*

Představte si, že svůj kód tesáte do kamene. Každé písmenko musíte pracně vyrazit do kamene, takže každé zbytečné písmenko je zbytečná práce navíc. Co pak ale vytesáte, prožije staletí.

Co dělá tento kód

```
1 "Python" is (not None)
2 ??
```

`is` zjišťuje, zda se jedná o stejný objekt.

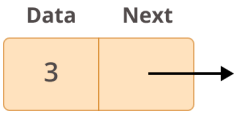
Tady nic nemá správný typ.

Opakování:

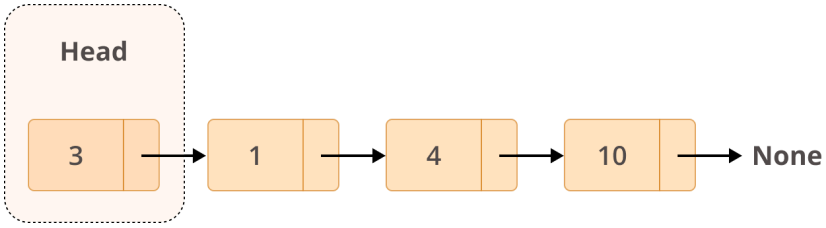
-

Lineární spojovaný seznam

"Převratný vynález": **spojení dat a strukturní informace**:

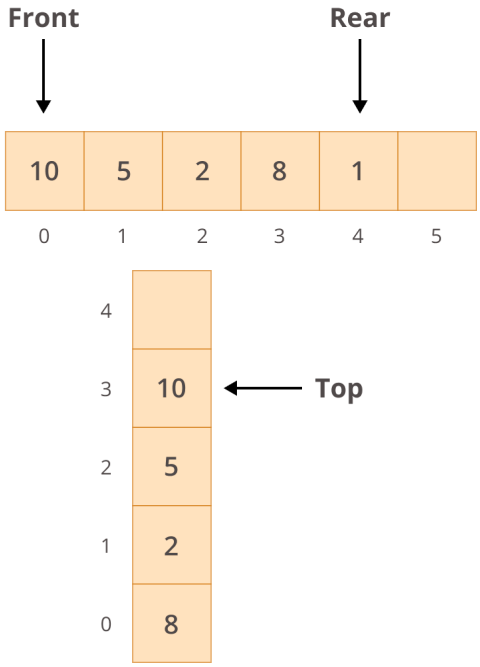


Takovéto jednotky pak umíme spojovat do větších struktur. LSS je nejjednodušší z nich.

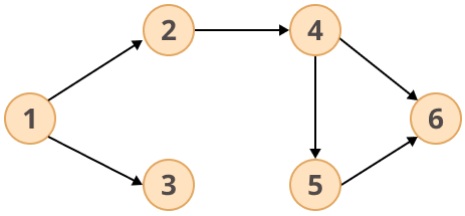


Aplikace:

- Fronty a zásobníky



- Grafy



Spojované seznamy v Pythonu

`list` v Pythonu je [dynamické pole]

- přidávání prvků: `insert` a `append`
- odebírání prvků: `pop` a `remove`

`collections.deque` je implementace fronty se dvěma konci.

- `append` / `appendleft`
- `pop` / `popleft`

Implementujeme spojovaný seznam

Spojovaný seznam s hlavou (kód v repozitáři, `code/Ex5/simply_linked_list1.py`)

```python
# Simple linked list


class Node:
    def __init__(self, value):
        """Polozku inicializujeme hodnotou value"""
        self.value = value
        self.next = None

    def __repr__(self):
        """Reprezentace objektu na Pythonovske konzoli"""
        return str(self.value)


class LinkedList:
    def __init__(self, values = None):
        """Spojovany seznam volitelne inicializujeme seznamem hodnot"""
        if values is None:
            self.head = None
            return
        self.head = Node(values.pop(0)) # pop vrati a odstrani hodnotu z values
        node = self.head
        for value in values:
            node.next = Node(value)
            node = node.next

    def __repr__(self):
        """Reprezentace na Pythonovske konzoli:
        Hodnoty spojene sipkami a na konci None"""
        values = []
        node = self.head
        while node is not None:
            values.append(str(node.value))
            node = node.next
        values.append("None")
        return " -> ".join(values)

    def __iter__(self):
        """Iterator prochazejici _hodnotami_ seznamu,
        napr. pro pouziti v cyklu for"""
        node = self.head
        while node is not None:
            yield node.value
            node = node.next

    def add_first(self, node):
        """Prida polozku na zacatek seznamu,
        tedy na head."""
        node.next = self.head
        self.head = node
```

```
51
52      def add_last(self, node):
53          """Prida polozku na konec seznamu."""
54          p = self.head
55          prev = None
56          while p is not None:
57              prev = p
58              p = p.next
59          prev.next = node
60
61
```

Vkládání a odstraňování prvků

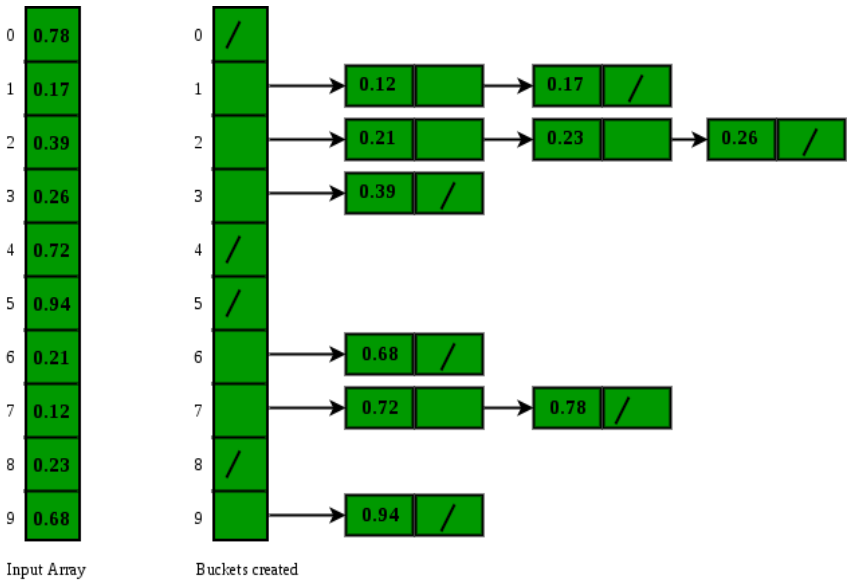
- `add_first`, `add_last`
- `add_before`, `add_after`
- `remove`

Třídění LSS

Utříděný seznam: `add` vloží prvek na správné místo

Jak utřídit již existující seznam?

Bucket sort vyžaduje složitou datovou strukturu

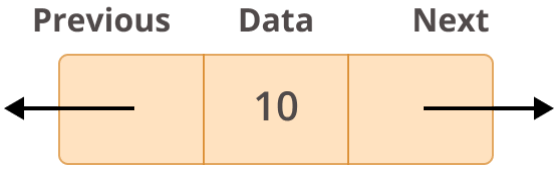


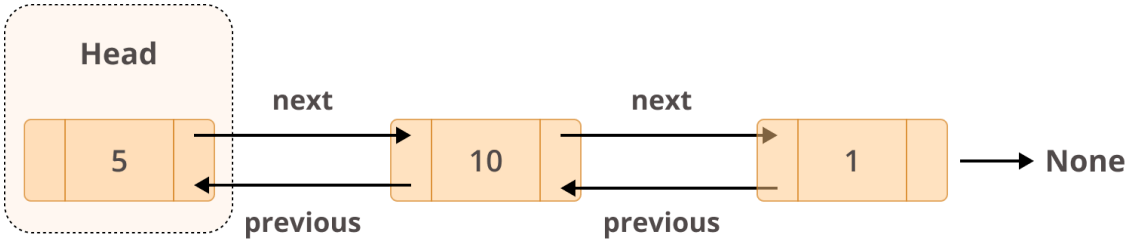
Insertion sort vyžaduje procházení ve dvou směrech.

Máme algoritmus, který by vystačil s průchody v jednom směru?

Varianty LSS

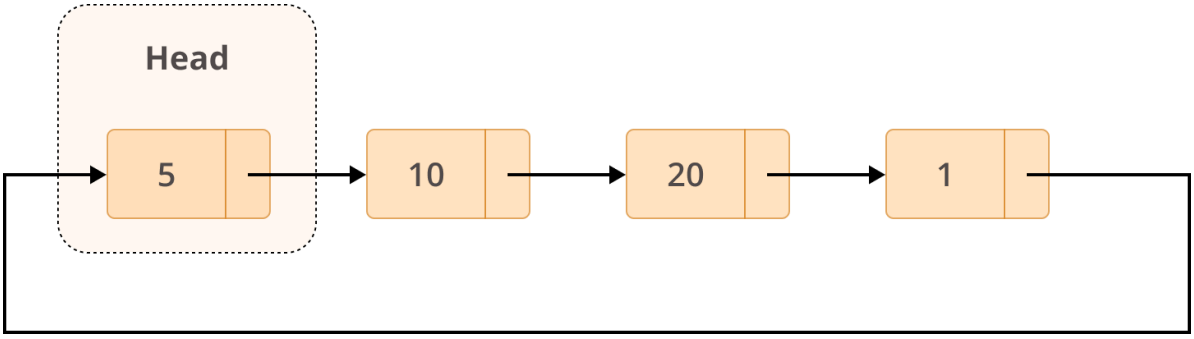
- **Dvojitě spojovaný seznam** - pro `deque`






```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.previous = None
```

- **Cyklický seznam**



Cyklickým seznamem můžeme procházet počínaje libovolným prvkem:

```
class CircularLinkedList:
    def __init__(self):
        self.head = None

    def traverse(self, starting_point=None):
        if starting_point is None:
            starting_point = self.head
        node = starting_point
        while node is not None and (node.next != starting_point):
            yield node
            node = node.next
        yield node

    def print_list(self, starting_point=None):
        nodes = []
        for node in self.traverse(starting_point):
            nodes.append(str(node))
        print(" -> ".join(nodes))
```

Jak to funguje:

```
>>> circular_llist = CircularLinkedList()
>>> circular_llist.print_list()
None

>>> a = Node("a")
>>> b = Node("b")
>>> c = Node("c")
>>> d = Node("d")
```

```
>>> a.next = b
>>> b.next = c
>>> c.next = d
>>> d.next = a
>>> circular_llist.head = a
>>> circular_llist.print_list()
a -> b -> c -> d

>>> circular_llist.print_list(b)
b -> c -> d -> a

>>> circular_llist.print_list(d)
d -> a -> b -> c
```