Základy algoritmizace

5. Spojové seznamy

doc. Ing. Jiří Vokřínek, Ph.D.
Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

Základy algoritmizace

- Dnes:
  - Spojový seznam

Abstraktní datový typ

Abstraktní datový typ jako objekt

- Objekt je instancí třídy
  - Třída je vzor – má rozhraní, data (vnitřní proměnné), metody (funkce)

Objektově orientované programování

```
class MyAbstractDataType:
    def __init__(self):
        self.dataOne = 1
        self.dataTwo = 2

    def operationOne(self, argument):
        self.dataOne = argument

    def operationTwo(self):
        return self.dataTwo

myData = MyAbstractDataType()
myData.operationOne(42)
myData.operationTwo()
```

Spojové struktury

Spojové struktury

- Seznam – List
  - Základní datová struktura pro uchovávání množiny prvků
  - Vložení/vyjmutí prvku může být velmi pomalé

- Spojové seznamy
  - Datová struktura realizující seznam dynamické délky
  - Každý prvek obsahuje
    - Datovou část (hodnota proměnné, objekt, …)
    - Referenci na další prvek

None v případě posledního prvku seznamu

  - První prvek seznamu se zpravidla označuje jako *head* nebo *start*

Realizováno jako referenční proměnná odkazující na první prvek seznamu

Spojové struktury

- Prvek seznamu – Node
- Základ i pro komplikovanější struktury

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

- Spojový seznam – LinkedList

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

Spojové struktury

- Test prázdnosti – isEmpty

```
def isEmpty(self):
    return self.head == None
```

- Přidání prvku – push

```
def push(self,item):
    node = Node(item)
    node.next = self.head
    self.head = node
```

Přidání prvku není závislé na počtu prvků v seznamu

Spojové struktury

- Odebrání prvního prvku – pop

```
def pop(self):
    if self.head:
        ret = self.head.data
        self.head = self.head.next
        return ret
```

Odebrání prvního prvku není závislé na počtu prvků v seznamu

- Zjištění počtu prvků – size

```
def size(self):
    current = self.head
    count = 0
    while current != None:
        count += 1
        current = current.next
    return count
```

Kompletní projití seznamu!

Spojové struktury

- Zrychlení operace size
  - Můžeme si pamatovat aktuální počet prvků *count*
  - Zvyšujeme a snižujeme při každé operaci vložení/vyjmutí

- Práce s posledním prvkem
  - Zavedeme si referenci na poslední prvek *end*
  - Pracujeme obdobně jako s *head*
    - V případě přidání prvku na začátek aktualizujeme pouze pokud byl seznam dosud prázdný
    - Aktualizujeme v případě přidání prvku na konec
    - Aktualizujeme při vyjmutí posledního prvku

Spojové struktury

```
class LinkedList:
    def __init__(self):
        self.head = None
        self.count = 0

    def push(self,item):
        node = Node(item)
        node.next = self.head
        self.head = node
        self.count +=1

    def pop(self):
        if self.head:
            ret = self.head.data
            self.head = self.head.next
            self.count -=1
            return ret

    def size(self):
        return self.count
```

Spojové struktury

```
class LinkedList:
    def __init__(self):
        self.head = None
        self.end = None
        self.count = 0

    def push(self,item):
        node = Node(item)
        if not self.head:
            self.head = self.end = node
        else:
            node.next = self.head
            self.head = node
        self.count +=1

    def pop(self):
        if self.head:
            ret = self.head.data
            self.head = self.head.next
            if self.head == None:
                self.end = None
            self.count -=1
            return ret
```

Spojové struktury

- Vložení prvku na konec – pushEnd

```
def pushEnd(self, item):
    node = Node(item)
    if not self.end:
        self.head = self.end = node
    else:
        self.end.next = node
        self.end = node
    self.count +=1
```

Spojové struktury

- Odebrání posledního prvku – popEnd

```
def popEnd(self):
    if self.head:
        self.count -=1
        ret = self.end.data
        if self.head == self.end:
            self.head = self.end = None
        else:
            cur = self.head
            while cur.next != self.end:
                cur = cur.next
            self.end = cur
            self.end.next = None
        return ret
```

Spojové struktury

- Obecně vkládání do seznamu
  - Na začátek – push
  - Na konec – pushEnd
  - Za daný prvek – potřebujeme referenci na prvek, za který chceme vkládat (node)

```
newNode = Node(item)
newNode.next = node.next
node.next = newNode
```

  - Na určitou pozici – insertAt
    - Nutno najít odpovídající prvek – průchod seznamu
    - Vložit za daný prvek
    - Ošetření mezních hodnot (velikost seznamu, head, end)

Spojové struktury

- Odebírání prvku
  - Ze začátku – pop
  - Z konce – popEnd
  - Daný prvek – potřebujeme referenci na prvek, který ho předchází (prevNode)

```
if prevNode.next == node:
    prevNode.next = node.next
    node.next = None
```

  - Z určité pozice – removeAt
    - Nutno najít odpovídající prvek – průchod seznamu
    - Odstranit prvek
    - Ošetření mezních hodnot (velikost seznamu, head, end)

Spojové struktury

- Složitější operace / práce s prvky
  - Vkládání a odebírání
  - Vyhledávání
  - Výpis prvků, …

Příklad: search a remove

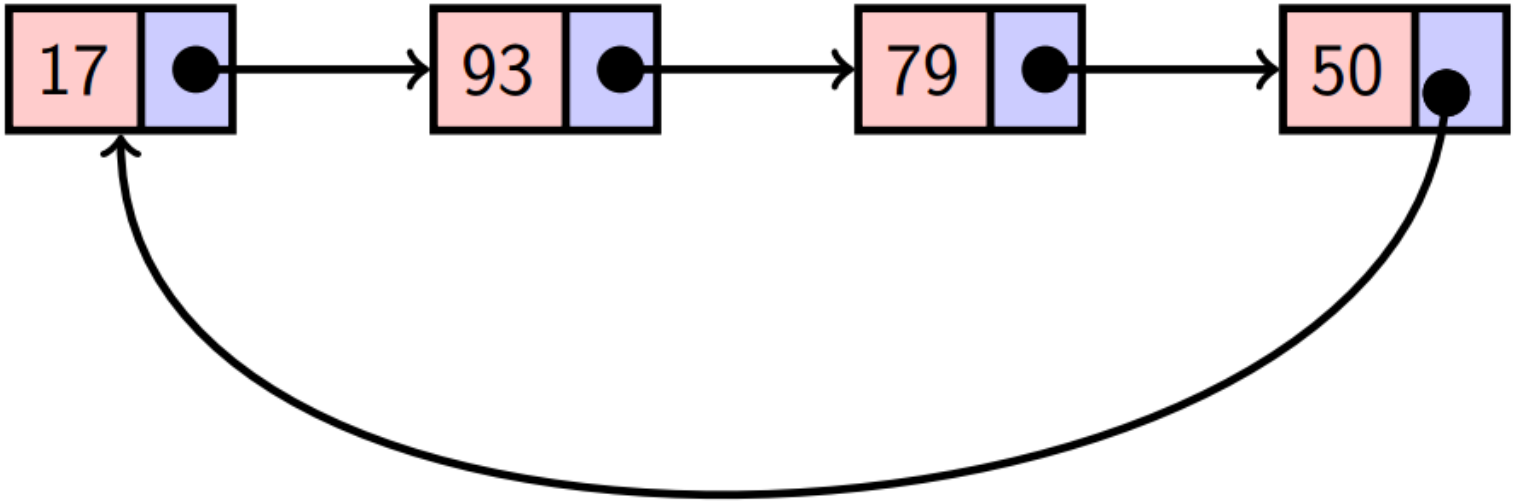
- Rozšíření základního spojového seznamu
  - Čítač prvků *count*
  - Reference na poslední prvek *end*

Zjednodušuje některé operace – z lineární náročnosti na konstantní

- Další možná rozšíření
  - Kruhový spojový seznam
  - Obousměrný spojový seznam

Spojové struktury

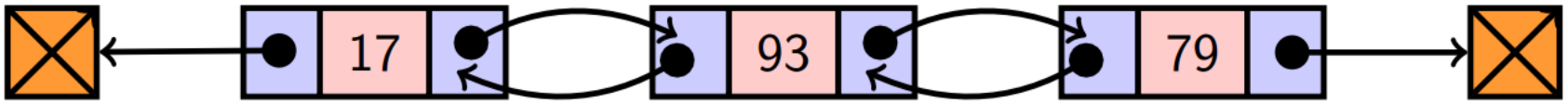
- Kruhový spojový seznam
  - Položka *next* posledního prvku odkazuje na první prvek
  - *head* a *end* ztrácejí smysl, ale je třeba držet odkaz na některý prvek (první, poslední přidaný, poslední vyhledaný, atp.)



Spojové struktury

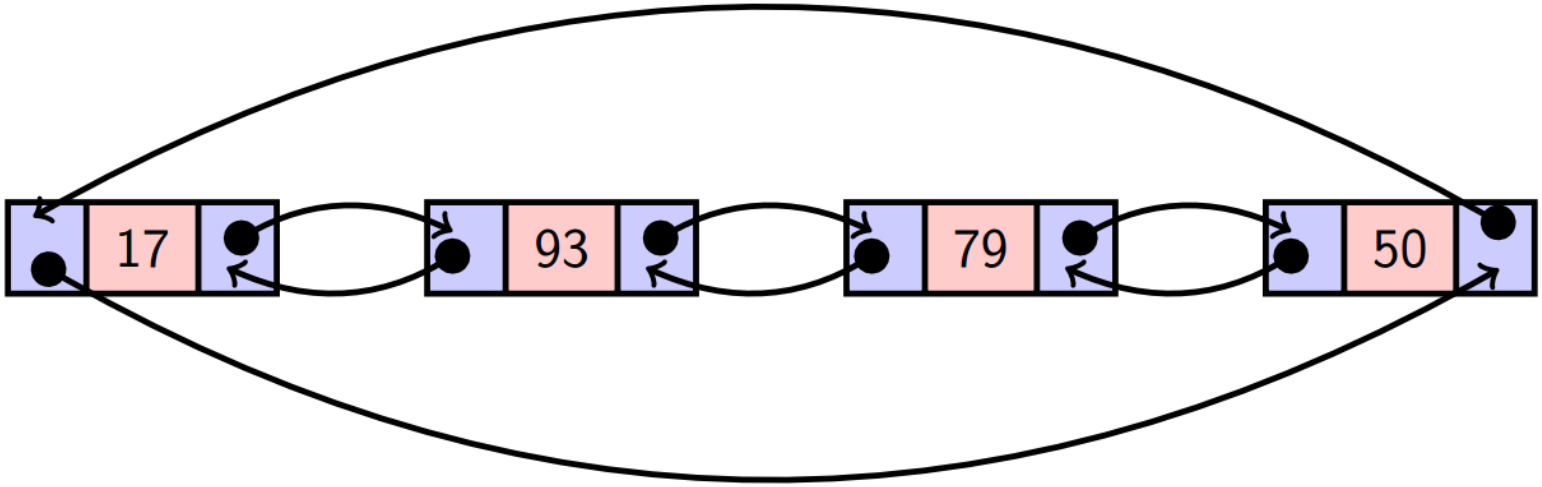
- Obousměrný spojový seznam
  - Každý prvek obsahuje odkaz na následující i předchozí položku v seznamu (*prev* a *next*)
  - První prvek má nastavenu položku *prev* na *None*
  - Poslední prvek má nastavenu položku *next* na *None*
  - Snadný „pohyb“ seznamem vpřed i vzad

Srovnejte popEnd u jednosměrného spojového seznamu



Spojové struktury

- Kruhový obousměrný spojový seznam
  - Kombinace předchozích



Základy algoritmizace

- Dnes:
  - Spojový seznam lineární, kruhový, obousměrný
- Zamyslete se
  - Jak implementovat obousměrný spojový seznam
  - Jak udržovat spojový seznam setříděný
  - Jak implementovat prioritní frontu spojovým seznamem

Příště rekurze

Základy algoritmizace

- Dodatečné zdroje:
  - Implementing a singly linked list in Python https://www.codefellows.org/blog/implementing-a-singly-linked-list-in-python
  - Linked lists http://www.cs.dartmouth.edu/~cs1/chapter19/index.html