Dynamické datové struktury I.

Seznam. Fronta. Zásobník.

Tomáš Bayer | bayertom@natur.cuni.cz

Katedra aplikované geoinformatiky a kartografie, Přírodovědecká fakulta UK.

Obsah přednášky

1. Odvozené datové struktury

Nazývány také dynamickými datovými strukturami.

Reprezentace tabulek, seznamů, grafů, matic, stromů.

Použití: rychlé třídění, rychlé vyhledávání, překlady výrazů, kompresní algoritmy, optimalizační úlohy, počítačová grafika.

Volba vhodné datové struktury klíčová pro efektivní řešení problému.

Optimalizace na úrovni datových struktur.

Aneb důležitý není pouze algoritmus...

Přehled dynamických datových struktur:

- Seznam (LIST)
- Zásobník (STACK)
- Fronta (QUEUE)
- Prioritní fronta (PRIORITY QUEUE)
- Strom (TREE)
- Tabulka (TABLE)

2. Seznam

Dynamická datová struktura, mění svoji velikost.

Uspořádaná/neuspořádaná posloupnost položek.

Každá položka obsahuje odkaz na následující/předchozí položku.

Patří mezi rekurzivní struktury, odkaz na položku stejného typu.

Vlastnost seznamu:

Rychlé přidávání prvků na počátek/konec seznamu ($O(1)$).

Vhodný pro procházení prvků sekvenčně.

Nevhodný pro přímý přístup k prvkům.

Typy seznamů:

- jednosměrný seznam (Single-Linked List),
- obousměrný seznam (Doubly-Linked List),
- kruhový seznam (Circular List).

3. Základní pojmy

Implementace:

Prostřednictvím pole.

Zřetězený seznam.

Procházení seznamem:

Procházením jedním směrem: Single-Linked List.

Procházení oběma směry: Doubly-Linked List.

Hlava (Head), ohon (Tail):

Head: první prvek seznamu.

Tail: poslední prvek v seznamu.

Odkaz na prvek:

Ukazuje na následující / předchozí prvek seznamu.

Optimalizace algoritmů:

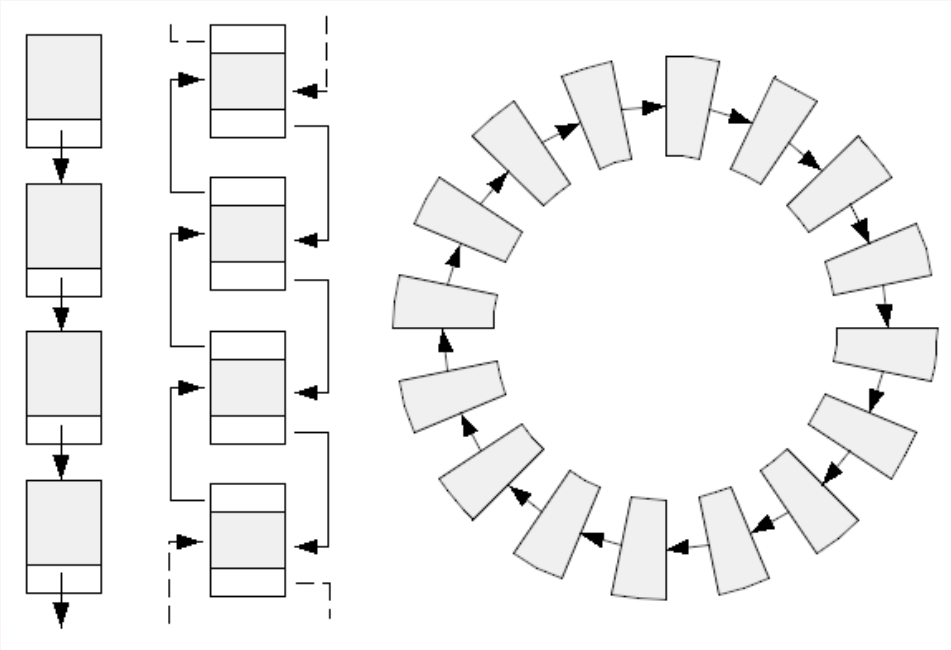
Volba algoritmu + volba vhodné dynamické struktury!

4. Ukázky seznamů.

Jednosměrný seznam: každý prvek seznamu odkazuje na předchozí resp. následující prvek seznamu. První resp. poslední prvek seznamu neodkazuje nikam (None).

Obousměrný seznam: každý prvek seznamu obsahuje odkaz na předchozí a následující prvek seznamu. První a poslední prvek seznamu neodkazují nikam (None).

Kruhový seznam: jednosměrný i obousměrný. Poslední prvek seznamu pak obsahuje odkaz na první prvek seznamu, resp. první prvek seznamu obsahuje odkaz na poslední prvek seznamu.



5. Operace se seznamem

Základní operace se seznamem:

- Přidání prvku na konec seznamu (PUSH): $O(1)$.
- Přidání prvku za daný prvek (INSERT): $O(1)$ + nalezení $O(N)$.
- Nalezení prvku (FIND): $O(N)$.
- Smazání prvku (DELETE): $O(1)$ + nalezení $O(N)$.
- Smazání seznamu (CLEAR): $O(N)$.

Vhodný při čtení/přidávání prvků na začátku/ konci seznamu.

Nevhodný pro přímý přístup.

Podpora pouze sekvenčního přístupu.

6. Jednosměrný seznam, ukázka

Třída Node, dvě datové položky:

`data`: uchovává hodnotu

`next`: odkaz na další uzel.

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

Při vytvoření uzlu neprovádíme jeho zařazení.

7. Vytvoření prázdného seznamu

Třída List, dvě datové položky: `first`, `last`.

Uchovávají odkaz na první a poslední uzel.

Umožňují přidání/mazání v čase $O(1)$.

Vytvoření prázdného seznamu:

1. Vytvoření odkazu na počáteční/koncový uzel seznamu.
2. Inicializace obou odkazů na `None`.

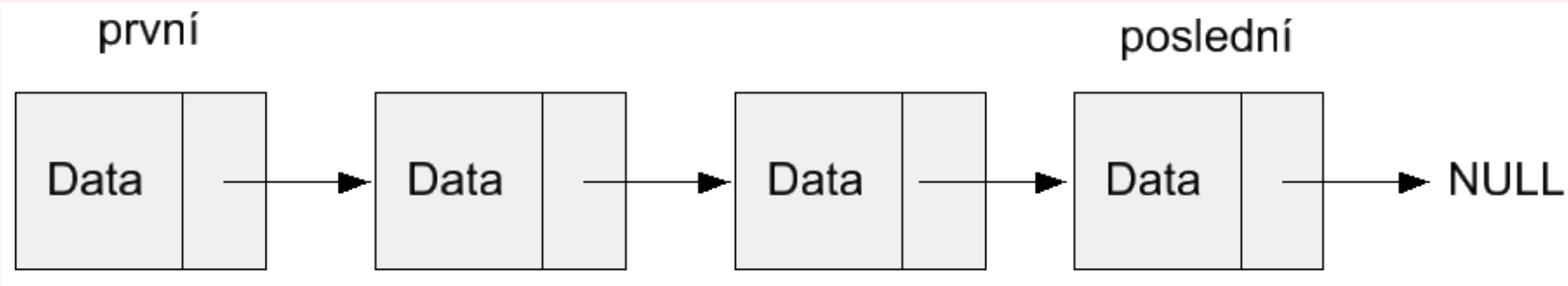
```
class LList:
    def __init__(self):
        self.first = None #Reference to first element
        self.last = None  #Reference to last element
```

8. Přidání prvku na konec seznamu

Přidání prvku na konec $O(1)$, na jiné místo $O(N)$.

Postup přidání prvku na konec seznamu:

1. Alokace nového uzlu *n*.
2. Nastavení parametrů uzlu:
   - *Prázdný seznam*
     První a poslední uzel seznamu *n*.
   - *Neprázdný seznam*
     Adresa přidávaného uzlu *n* uložena do položky `next`.
     Uzel *n* se stává posledním uzlem.



9. Přidání prvku na konec seznamu, ukázka

```
def push(self, data):
    n = Node(data)
    if self.first == None:      #List is empty
        self.first = n          #New node is first and last
        self.last = n
    else:
        self.last.next = n      #Link to the previous node
        self.last = n           #Node n becomes the last
```

10. Výpis jednosměrného seznamu

Prováděn sekvenčně, od prvního uzlu k poslednímu uzlu.

Složitost $O(n)$.

Postup výpisu prvků:

1. Získání adresy prvního uzlu `n` (hlava).
2. Dokud nedosáhneme konce seznamu:
   - Výpis hodnoty aktuálního uzlu `n`.
   - Inkrementace uzlu `n=n.next`.

```
def Print(self):
    n = self.first      #Adresa prvniho uzlu
    while n != None:    #Dokuf
       print (n.data)
       n = n.next
```

11. Výpis jednosměrného seznamu, ukázka

```
p = LList()
l.push("Monday")
l.push("Tuesday")
l.push("Wednesday")
l.push("Thursday")
l.Print()
```

Pak:

```
Monday
Tuesday
Wednesday
Thursday
Monday
```

12. Nalezení prvku v jednosměrném seznamu

Prohledávání prováděno sekvenčně.

Složitost $O(n)$.

Postup nalezení prvku s hodnotou *h*:

1. Získání adresy prvního uzlu `n`.
2. Dokud nedosáhneme konce seznamu:
   - Pokud `n.data == h`, vrať `n`.
   - Jinak inkrementace uzlu `n=n.next`.
3. Hodnota *h* není v seznamu, vrať `None`.

13. Nalezení prvku v jednosměrném seznamu, ukázka

```
def find(self, h):
    n = self.first
    while n != None:
        if n.data == h:
            return n
        n = n.next
    return None
```

14. Přidání prvku za daný prvek

Složitost operace $O(1)$.

Hledání uzlu, za který přidáváme $O(N)$.

Nutno sekvenčně projít předcházejících $n - 1$ prvků.

Postup přidání uzlu n za uzel m:

1. Alokace nového uzlu n.
2. Zařazení uzlu:
   - Propojení uzlu n s následníkem m
     `n.next = m.next;`
   - Propojení uzlu m s uzlem n
     `m.next = n;`

```
def insert(self, m, data):
    n = Node(data)
    n.next = m.next
    m.next = n
```

15. Smazání prvku v jednosměrném seznamu

Vymazání prvku zadaného adresou ze seznamu, složitost $O(1)$.

Problém: neznáme prvek předcházející hledanému prvku.

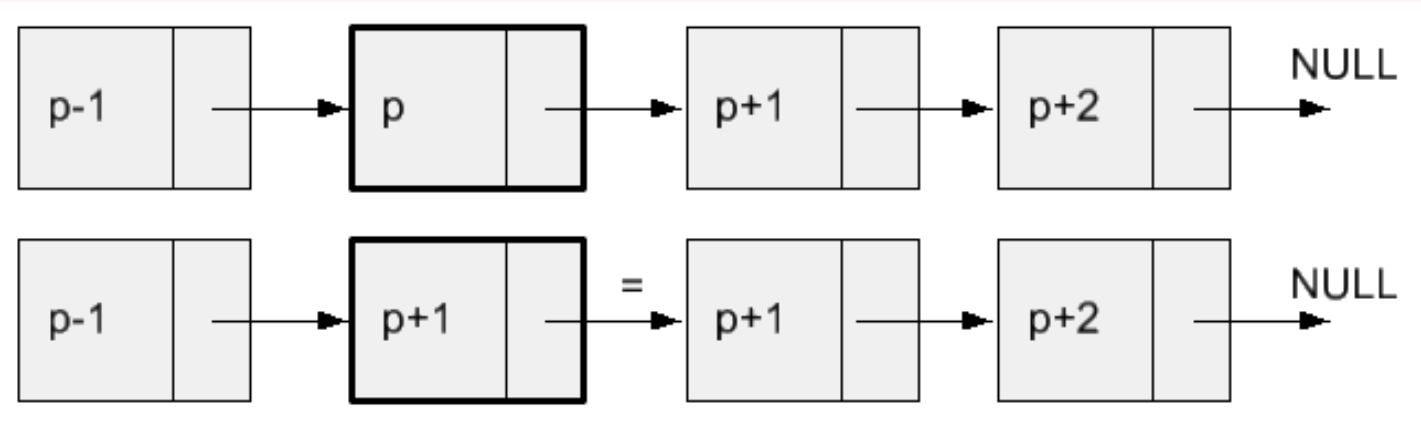
Do mazaného prvku zkopírujeme obsah následujícího prvku (2 stejné prvky).

Smažeme prvek za mazaným prvkem.

Postup smazání prvku:

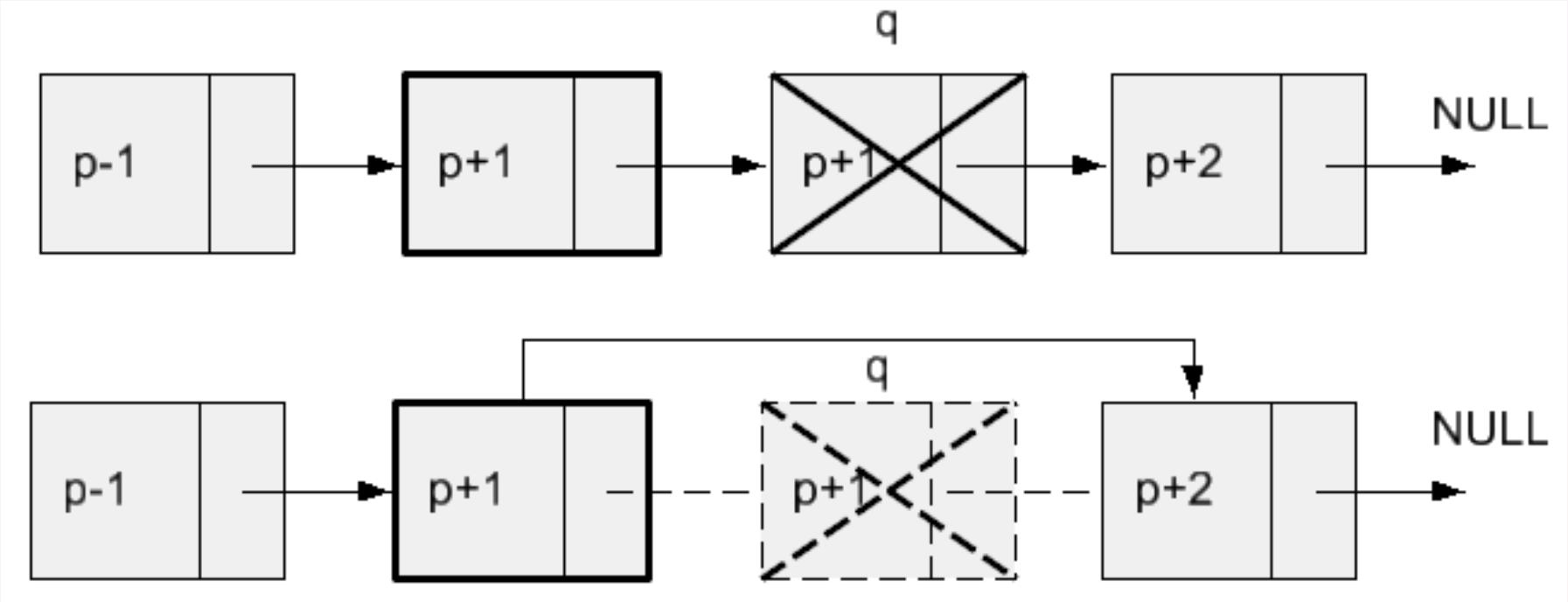
1. Přiřazení obsahu následujícího prvku `n.data = n.next.data`.
2. Smazání prvku `n.next` ve 3 krocích:
   - Pomocný uzel `m = n.next`.
   - Nastavení následníka: `n.next = m.next`.
   - Smazání uzlu `m`.

Fáze 1: `n=n.dalsi`. Dva stejné prvky za sebou.



16. Smazání prvku v jednosměrném seznamu

Fáze 2: Nastavení následníka mazaného prvku na jeho následníka.



Fáze 3: Smazání druhého z dvojice stejných prvků.

17. Smazání prvku v jednosměrném seznamu, ukázka

Jednoduchá implementace:

```
def delete(self, n):
    n.data = n.next.data
    m = n.next
    n.next = n.next.next
    m = None
```

Nelze použít pro poslední prvek (nemá následníka).

Nutné sekvenční procházení, složitost $O(n)$.

Preferován Doubly Linked List.

18. Zásobník (Stack)

Princip zásobníku:

Reprezentuje model LIFO (Last In - First Out).

Odebíráme data v opačném pořadí, než v jakém jsme je do něj uložili.

Pracujeme pouze s prvkem na vrcholu zásobníku.

V běžném životě model zásobníku např. batoh.

Vyndaváme z něj věci v opačném pořadí, než je do něj ukládáme.

Použití při zpracovávání dat v sekvenčním pořadí.

Implementování zásobníku:

- Spojovým seznamem (Doubly Linked List).
- Polem.

Složitost operací $O(1)$.

19. Základní pojmy a operace se zásobníkem

Dno zásobníku:

Nejspodnější prvek v zásobníku.

Přidán do zásobníku jako první.

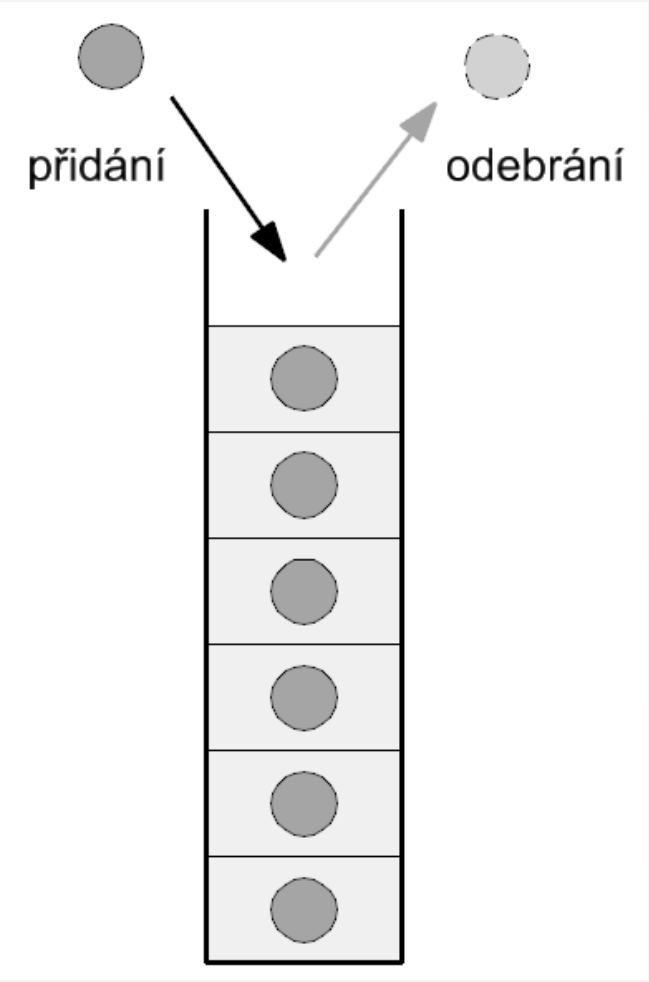
Vrchol zásobníku:

Nejvrchnější prvek v zásobníku.

Přidán do zásobníku jako poslední.

Operace se zásobníkem:

- Vytvoření zásobníku.
- Přidání prvku do zásobníku (PUSH).
- Odebrání prvku ze zásobníku (POP).



20. Implementace zásobníku, třída Node

Implementace zásobníku s použitím obousměrného lineárního seznamu.

Každá položka seznamu obsahuje odkaz na:

- předchozí položku seznamu,
- následující položku seznamu.

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

21. Vytvoření prázdného zásobníku, ukázka

Postup vytvoření prázdného zásobníku:

1. Vytvoření odkazu na počáteční a koncový uzel seznamu.
2. Inicializace obou odkazů na `None`.

```
class Stack:
    def __init__(self):
        self.first = None
        self.last = None
```

22. Přidání uzlu do zásobníku: PUSH

Princip přidání uzlu do zásobníku:

Přidávaný uzel je umisťován vždy na konec zásobníku.

Efektivnější varianta než jeho přidání na počátek.

Nutno nastavit přidávaný uzel jako následníka posledního uzlu a tomuto uzlu nastavit předchůdce.

Postup přidání uzlu do zásobníku:

1. Alokace nového uzlu *n*.
2. Nastavení parametrů uzlu:
   - *Prázdný seznam*
     První a poslední uzel seznamu *n*.
   - *Neprázdný seznam*
     Uzel *n* se stává nástupcem posledního.
     Jeho předchůdcem původní poslední.
     Uzel `n` se stává posledním uzlem.

23. Přidání uzlu do zásobníku, ukázka

```
def push(self, data):
    n = Node(data)
    if self.first == None:
        self.first = n
        self.last = n
    else:
        self.last.next = n
        n.prev = self.last
        self.last = n
```

24. Vyjmutí uzlu ze zásobníku

Ze zásobníku vždy vyjímán poslední uzel.

Nutno nastavit nový konec zásobníku a jeho následníka nasměrovat na `None`.

Postup vyjmutí uzlu ze zásobníku:

1. Pokud je zásobník prázdný, skonči.
2. Zapamatování posledního uzlu: `n = last`
3. Pokud S tvořen jedním prvkem, vznikne prázdný S.
4. Pokud S není prázdný:
   - Nastavení posledního uzlu: `last = last.prev`
   - Nastavení následníka: `last.next = None`
5. Smazání uzlu `n`: `n = None`

25. Vyjmutí uzlu ze zásobníku, ukázka

```
def pop(self):
    if self.first == None:      #Empty stack
        return;
    n = self.last
    if self.last.prev == None: #1 element in S
        self.first = None
        self.last = None
    else:                       #More elements in S
        self.last = n.prev      #Assign the predecessor
        self.last.next = None   #Reset nex element
    n = None                    #Delete node
```

26. Ukázka výpisu obsahu zásobníku

```
S = Stack();
S.push("Monday")
S.push("Tuesday")
S.push("Wednesday")
S.push("Thursday")
S.pop();
S.Print();
```

Výsledek:

```
>> Monday
>> Tuesday
>> Wednesday
```

27. Fronta (Query)

Reprezentuje model FIFO (First In-First Out).

Z fronty odebíráme data ve stejném pořadí, v jakém jsem je do ní uložili.

Pracovat lze pouze s prvkem, který je na čele fronty.

Využití při sekvenčním zpracování dat.

Ve frontě nemůžeme předbíhat.

Existuje i fronta s předbíháním, tzv. prioritní fronta.

Implementace fronty:

- Polem (Doubly Linked List).
- Spojovým seznamem.

Složitost operací $O(1)$.

28. Základní pojmy a operace s frontou

Konec fronty:

Prvek na poslední pozici ve frontě.

Do fronty frony přidán jako poslední.

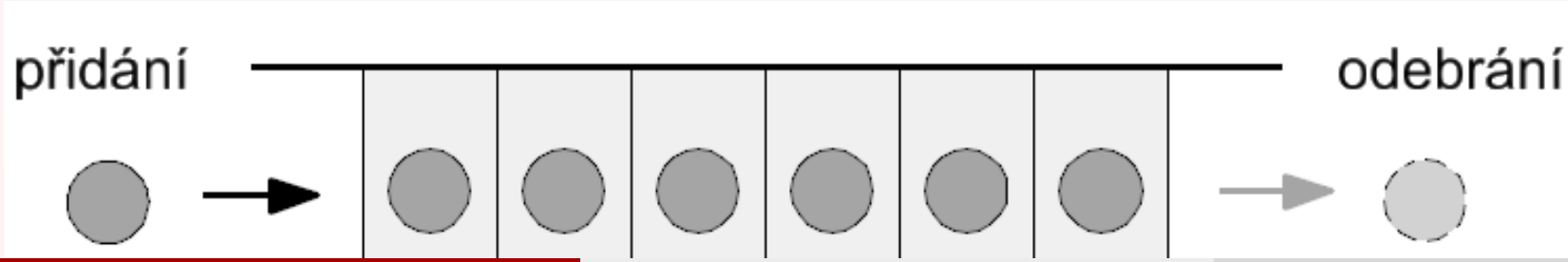
Čelo fronty:

Prvek na první pozici ve frontě,

Do fronty přidán jako první.

Operace s frontou:

- Vytvoření fronty.
- Přidání prvku do fronty (PUSH).
- Odebrání prvku z fronty (POP).



29. Implementace fronty-třída Node

Implementace fronty s použitím obousměrného lineárního seznamu.

Každá položka seznamu obsahuje odkaz na:

- předchozí položku seznamu,
- následující položku seznamu.

Analogie se zásobníkem.

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

30. Vytvoření prázdné fronty, ukázka

Postup vytvoření prázdné fronty (analogie zásobníku):

1. Vytvoření odkazu na počáteční a koncový uzel seznamu.
2. Inicializace obou odkazů na `None`.

```
class Query:
    def __init__(self):
        self.first = None
        self.last = None
```

31. Přidání uzlu do fronty: PUSH

Stejný princip jako u zásobníku.

Přidávaný uzel je umisťován vždy na konec seznamu.

Nutno nastavit přidávaný uzel jako následníka posledního uzlu a tomuto uzlu nastavit předchůdce.

Postup stejný jako u zásobníku:

1. Alokace nového uzlu *n*.
2. Nastavení parametrů uzlu:
   - *Prázdný seznam*
     První a poslední uzel seznamu *n*.
   - *Neprázdný seznam*
     Uzel *n* se stává nástupcem posledního.
     Jeho předchůdcem původní poslední.
     Uzel `n` se stává posledním uzlem.

32. Přidání uzlu do fronty: PUSH, ukázka

```
def push(self, data):
    n = Node(data)
    if self.first == None:
        self.first = n
        self.last = n
    else:
        self.last.next = n
        n.prev = self.last
        self.last = n
```

33. Vyjmutí uzlu z fronty: POP

Z fronty vždy vyjímán první uzel, ze zásobníku poslední.

Nutno nastavit nový počáteční uzel fronty a jeho předchůdce.

Postup vyjmutí uzlu z fronty:

1. Pokud je fronta prázdná, skonči.
2. Zapamatování prvního uzlu: `n = first`
3. Pokud Q tvořená jedním prvkem, vznikne prázdná Q.
4. Pokud Q není prázdná:
   - Nastavení prvního uzlu: `first = first.next`
   - Reset předchůdce: `first.prev = None`
5. Smazání uzlu `n`: `n = None`

34. Vyjmutí prvku z fronty: POP, ukázka

```
def pop(self):
    n = self.first
    if self.first == None:                  #Empty Q
        return;
    if self.last.prev == None:              #1 element in Q
        self.first = None
        self.last = None
    else:                                   #More elements in Q
        self.first = self.first.next        #Assign successor
        self.first.prev = None              #Reset link
    n = None                                #Delete node
```

35. Ukázka výpisu obsahu fronty

```
Q = Query();
Q.push("Monday")
Q.push("Tuesday")
Q.push("Wednesday")
Q.push("Thursday")
Q.pop()
Q.Print()
```

Výsledek:

```
>> Tuesday
>> Wednesday
>> Thursday
```