Složitost implementací abstraktních datových typů

KIV/ADT – 4. přednáška

Miloslav Konopík, Libor Váša

7. března 2024

Obsah

. . . aneb malý výlet do KIV/IDT.

Dynamické pole

Pole

Pole je uspořádaná, měnitelná kolekce prvků s duplicitami s pevnou délkou.

- Nízkoúrovňová datová struktura.
- V Pythonu neexistuje přímo, ale je na pozadí struktry `list`.
- Přístup na index $\Theta(1)$.
  - Lineární vztah mezi indexem a adresou v paměti.

index	1	2	3	4	5	6	7	8
pole	9	1	17	9	12	13	9	3
paměť	0x404	0x408	0x40c	0x410	0x414	0x418	0x41c	0x420

Změna velikosti pole

- Při změně velikost pole může být nutné alokovat novou paměť a data kopírovat.
- Operační systém může podporovat změnu velikosti i bez nutnosti kopírování dat.
- Jaká je optimální strategie na zvětšování pole?

Změna velikosti pole

- Při změně velikost pole může být nutné alokovat novou paměť a data kopírovat.
- Operační systém může podporovat změnu velikosti i bez nutnosti kopírování dat.
- Jaká je optimální strategie na zvětšování pole?
- Z [PPA] zvětšujeme o násobek 2x. Proč? Dále...

index	1	2	3	4
pole	9	1	17	9
paměť	0x404	0x408	0x40c	0x410

index	1	2	3	4	5	6	7	8
pole	9	1	17	9	12	·	·	·
paměť	0x404	0x408	0x40c	0x410	0x414	0x418	0x41c	0x420

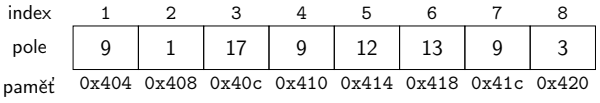
Změna velikosti pole

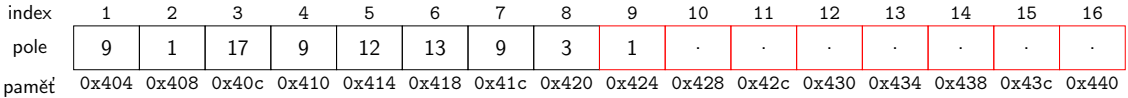
- Při změně velikost pole může být nutné alokovat novou paměť a data kopírovat.
- Operační systém může podporovat změnu velikosti i bez nutnosti kopírování dat.
- Jaká je optimální strategie na zvětšování pole?
- Z [PPA] zvětšujeme o násobek 2x. Proč? Dále...

index	1	2	3	4	5	6	7	8
pole	9	1	17	9	12	13	9	3
paměť	0x404	0x408	0x40c	0x410	0x414	0x418	0x41c	0x420

Změna velikosti pole

- Při změně velikost pole může být nutné alokovat novou paměť a data kopírovat.
- Operační systém může podporovat změnu velikosti i bez nutnosti kopírování dat.
- Jaká je optimální strategie na zvětšování pole?
- Z [PPA] zvětšujeme o násobek 2x. Proč? Dále...





Amortizovaná složitost

Amortizovaná složitost

Amortizovaná složitost:

- Amortizovaná analýza: metoda pro analýzu výkonu algoritmu průměrovaného přes více operací [4]
- Motivace: vyhnout se pesimismu nejhoršího případu a sledovat průběh časové funkce $T(n)$ pro všechny hodnoty n.
- Idea: rozložit náklady na drahé operace přes několik levnějších operací
- Techniky [2] : agregovaná analýza [5], účetní metoda [1], metoda výpočtu potenciálu [3]

Amortizovaná složitost zvětšování pole

- třída složitosti v průměrném případě
- zkoumáme přidání n prvků, pro jednoduchost začneme s polem délky 1
- kolikrát se pole zvětšuje?
  - záleží na n
  - $n <= 1$:

Amortizovaná složitost zvětšování pole

- třída složitosti v průměrném případě
- zkoumáme přidání n prvků, pro jednoduchost začneme s polem délky 1
- kolikrát se pole zvětšuje?
  - záleží na n
  - $n <= 1 : 0\times$

Amortizovaná složitost zvětšování pole

- třída složitosti v průměrném případě
- zkoumáme přidání n prvků, pro jednoduchost začneme s polem délky 1
- kolikrát se pole zvětšuje?
  - záleží na n
  - $n <= 1 : 0\times$
  - $n <= 2 :$

Amortizovaná složitost zvětšování pole

- třída složitosti v průměrném případě
- zkoumáme přidání n prvků, pro jednoduchost začneme s polem délky 1
- kolikrát se pole zvětšuje?
  - záleží na n
  - $n <= 1 : 0\times$
  - $n <= 2 : 1\times$

Amortizovaná složitost zvětšování pole

- třída složitosti v průměrném případě
- zkoumáme přidání n prvků, pro jednoduchost začneme s polem délky 1
- kolikrát se pole zvětšuje?
  - záleží na n
  - $n <= 1 : 0\times$
  - $n <= 2 : 1\times$
  - $n <= 4 :$

Amortizovaná složitost zvětšování pole

- třída složitosti v průměrném případě
- zkoumáme přidání n prvků, pro jednoduchost začneme s polem délky 1
- kolikrát se pole zvětšuje?
  - záleží na n
  - $n <= 1 : 0\times$
  - $n <= 2 : 1\times$
  - $n <= 4 : 2\times$

Amortizovaná složitost zvětšování pole

- třída složitosti v průměrném případě
- zkoumáme přidání n prvků, pro jednoduchost začneme s polem délky 1
- kolikrát se pole zvětšuje?
  - záleží na n
  - $n <= 1 : 0\times$
  - $n <= 2 : 1\times$
  - $n <= 4 : 2\times$
  - $n <= 8 : 3\times$

Amortizovaná složitost zvětšování pole

- třída složitosti v průměrném případě
- zkoumáme přidání n prvků, pro jednoduchost začneme s polem délky 1
- kolikrát se pole zvětšuje?
  - záleží na n
  - $n <= 1 : 0\times$
  - $n <= 2 : 1\times$
  - $n <= 4 : 2\times$
  - $n <= 8 : 3\times$
  - $n <= 2^k : k\times$

Amortizovaná složitost zvětšování pole

- třída složitosti v průměrném případě
- zkoumáme přidání n prvků, pro jednoduchost začneme s polem délky 1
- kolikrát se pole zvětšuje?
  - záleží na n
  - $n <= 1 : 0\times$
  - $n <= 2 : 1\times$
  - $n <= 4 : 2\times$
  - $n <= 8 : 3\times$
  - $n <= 2^k : k\times$
- počet zvětšování: $k = \lceil log_2(n) \rceil$

Amortizovaná složitost zvětšování pole

- třída složitosti v průměrném případě
- zkoumáme přidání n prvků, pro jednoduchost začneme s polem délky 1
- kolikrát se pole zvětšuje?
  - záleží na n
  - $n <= 1 : 0\times$
  - $n <= 2 : 1\times$
  - $n <= 4 : 2\times$
  - $n <= 8 : 3\times$
  - $n <= 2^k : k\times$
- počet zvětšování: $k = \lceil log_2(n) \rceil$
- počet kopírovaných prvků pro i-té zvětšení: $2^{(i-1)}$

Amortizovaná složitost zvětšování pole

- třída složitosti v průměrném případě
- zkoumáme přidání n prvků, pro jednoduchost začneme s polem délky 1
- kolikrát se pole zvětšuje?
  - záleží na n
  - $n <= 1 : 0\times$
  - $n <= 2 : 1\times$
  - $n <= 4 : 2\times$
  - $n <= 8 : 3\times$
  - $n <= 2^k : k\times$
- počet zvětšování: $k = \lceil log_2(n) \rceil$
- počet kopírovaných prvků pro i-té zvětšení: $2^{(i-1)}$
- celkový počet operací při zvětšování: $\sum_{i=1}^{\lceil log_2(n) \rceil} 2^{(i-1)}$

Počet operací při zvětšení pole

Data:

Celkový počet operací kopírování:

Počet operací při zvětšení pole

Data:

Celkový počet operací kopírování:

Počet operací při zvětšení pole

Data:

Celkový počet operací kopírování:

Počet operací při zvětšení pole

Data:

Celkový počet operací kopírování:

Počet operací při zvětšení pole

Data:

Celkový počet operací kopírování:

Počet operací při zvětšení pole

Data:

Celkový počet operací kopírování:

Počet operací při zvětšení pole

Data:

Celkový počet operací kopírování:

Počet operací při zvětšení pole

Data:

Celkový počet operací kopírování:

Počet operací při zvětšení pole

Data:

Celkový počet operací kopírování:

Počet operací při zvětšení pole

Data:

Celkový počet operací kopírování:

Počet operací při zvětšení pole

Data:

Celkový počet operací kopírování:

Počet operací při zvětšení pole

Data:

Celkový počet operací kopírování:

Celkový počet operací při zvětšení

počet prvků n	počet zvětšení	počet kopírovaných prvků o
$n <= 1$	$0\times$	0
$n <= 2$	$1\times$	1

Celkový počet operací při zvětšení

počet prvků n	počet zvětšení	počet kopírovaných prvků o
$n <= 1$	$0\times$	0
$n <= 2$	$1\times$	1
$n <= 4$	$2\times$	$1 + 2 = 3$

Celkový počet operací při zvětšení

počet prvků n	počet zvětšení	počet kopírovaných prvků o
$n <= 1$	$0\times$	0
$n <= 2$	$1\times$	1
$n <= 4$	$2\times$	$1 + 2 = 3$
$n <= 8$	$3\times$	$1 + 2 + 4 = 7$

Celkový počet operací při zvětšení

počet prvků n	počet zvětšení	počet kopírovaných prvků o
$n <= 1$	$0\times$	0
$n <= 2$	$1\times$	1
$n <= 4$	$2\times$	$1 + 2 = 3$
$n <= 8$	$3\times$	$1 + 2 + 4 = 7$
$n <= 16$	$4\times$	$1 + 2 + 4 + 8 = 15$

Celkový počet operací při zvětšení

počet prvků n	počet zvětšení	počet kopírovaných prvků o
$n <= 1$	$0\times$	0
$n <= 2$	$1\times$	1
$n <= 4$	$2\times$	$1 + 2 = 3$
$n <= 8$	$3\times$	$1 + 2 + 4 = 7$
$n <= 16$	$4\times$	$1 + 2 + 4 + 8 = 15$
$n <= 2^k$	$k\times$	$2^k - 1$

Celkový počet operací při zvětšení

počet prvků n	počet zvětšení	počet kopírovaných prvků o
$n <= 1$	$0\times$	0
$n <= 2$	$1\times$	1
$n <= 4$	$2\times$	$1 + 2 = 3$
$n <= 8$	$3\times$	$1 + 2 + 4 = 7$
$n <= 16$	$4\times$	$1 + 2 + 4 + 8 = 15$
$n <= 2^k$	$k\times$	$2^k - 1$

Nejhorší případ: hned po zvětšení pole:

$n = 2^k + 1, o = 2^{k+1} - 1$

Celkový počet operací při zvětšení

počet prvků n	počet zvětšení	počet kopírovaných prvků o
$n <= 1$	$0\times$	0
$n <= 2$	$1\times$	1
$n <= 4$	$2\times$	$1+2=3$
$n <= 8$	$3\times$	$1+2+4=7$
$n <= 16$	$4\times$	$1+2+4+8=15$
$n <= 2^k$	$k\times$	2^k-1

Nejhorší případ: hned po zvětšení pole:

$n = 2^k + 1, o = 2^{k+1} - 1$
$o = 2(2^k) - 1$

Celkový počet operací při zvětšení

počet prvků n	počet zvětšení	počet kopírovaných prvků o
$n <= 1$	$0\times$	0
$n <= 2$	$1\times$	1
$n <= 4$	$2\times$	$1+2=3$
$n <= 8$	$3\times$	$1+2+4=7$
$n <= 16$	$4\times$	$1+2+4+8=15$
$n <= 2^k$	$k\times$	2^k-1

Nejhorší případ: hned po zvětšení pole:

$n = 2^k + 1, o = 2^{k+1} - 1$

$o = 2(2^k) - 1 = 2(2^k + 1) - 3$

Celkový počet operací při zvětšení

počet prvků n	počet zvětšení	počet kopírovaných prvků o
$n <= 1$	$0\times$	0
$n <= 2$	$1\times$	1
$n <= 4$	$2\times$	$1+2=3$
$n <= 8$	$3\times$	$1+2+4=7$
$n <= 16$	$4\times$	$1+2+4+8=15$
$n <= 2^k$	$k\times$	2^k-1

Nejhorší případ: hned po zvětšení pole:

$n = 2^k + 1, o = 2^{k+1} - 1$

$o = 2(2^k) - 1 = 2(2^k + 1) - 3 = 2n - 3 \in \Theta(n)$

Celkový počet operací při zvětšení

počet prvků n	počet zvětšení	počet kopírovaných prvků o
$n <= 1$	$0\times$	0
$n <= 2$	$1\times$	1
$n <= 4$	$2\times$	$1+2=3$
$n <= 8$	$3\times$	$1+2+4=7$
$n <= 16$	$4\times$	$1+2+4+8=15$
$n <= 2^k$	$k\times$	2^k-1

Nejhorší případ: hned po zvětšení pole:

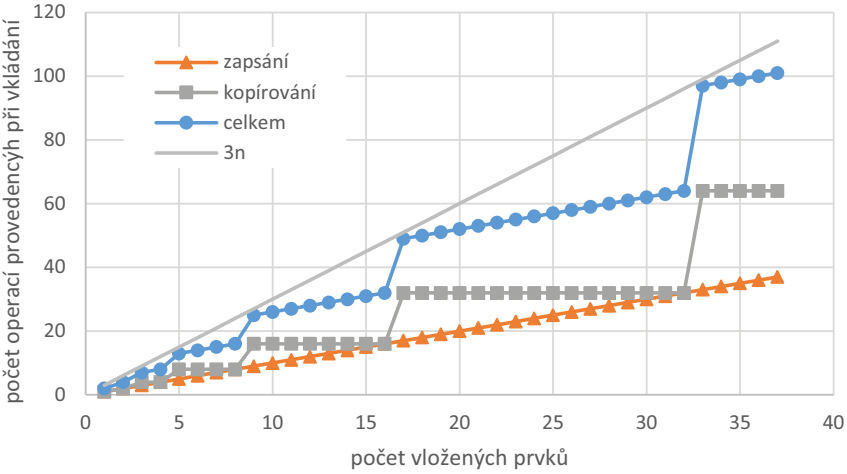
$n = 2^k + 1, o = 2^{k+1} - 1$

$o = 2(2^k) - 1 = 2(2^k + 1) - 3 = 2n - 3 \in \Theta(n)$

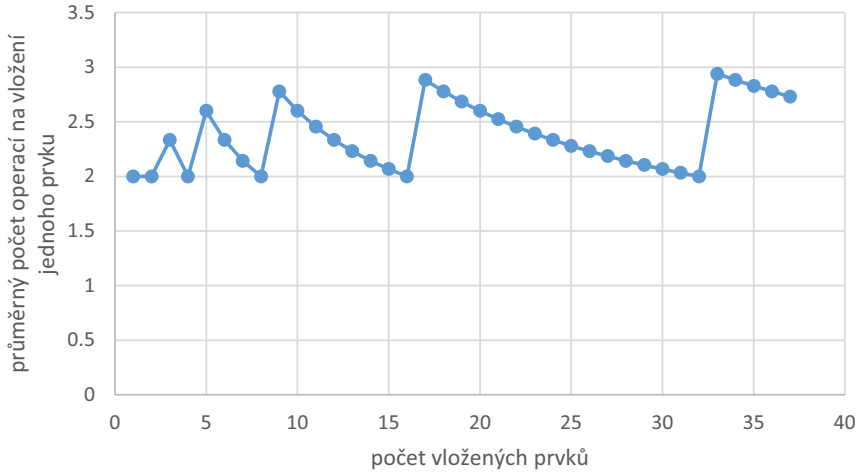
Závěr

- složitost operace "přidání n prvků" je $\mathcal{O}(n)$
- amortizovaná složitost operace "přidání jednoho prvku" je $\Theta(1)$

Celkové počty operací při přidávání



Amortizovaný počet operací při přidávání



Amortizovaná složitost

Důležitá poznámka

- tato analýza platí tehdy, když při zvětšování alokujeme dvojnásobné pole
- platila by i při použití jiného násobku (1.5x namísto 2x)

Amortizovaná složitost

Důležitá poznámka

- tato analýza platí tehdy, když při zvětšování alokujeme dvojnásobné pole
- platila by i při použití jiného násobku (1.5x namísto 2x)

- přizvětšování o konstantu analýza neplatí!
  - přidání n prvků je v takovém případě $\Theta(n^2)$,
  - přidání jednoho prvku je pak v průměru $\Theta(n)$,
  - pro zvětšování o jeden prvek je to zjevné,
  - zvětšování o větší počty (o 1000...) nepomůže!

Kvíz I

Implementace polem

Seznam implementovaný polem

Seznam je uspořádaná, měnitelná kolekce prvků s duplicitami.

Podporované operace:

- umožňuje vložení prvku na jakoukoli pozici,
- umožňuje vybrání / odebrání prvku z jakékoli pozice,
- procházení seznamu.

Seznam – vložení a odebrání prvku na pozici

Kroky vložení do seznamu l o velikost n na pozici k:

1. Kontrola velikosti pole: pokud $n + 1 > 2^k$ ($k = \lceil log_2(n) \rceil$), zvětšíme.
2. Posun všech prvků: $l[i + 1] = l[i]$ pro $i = n - 2 \ldots k$.
3. Přepsání prvku na pozici k.

Seznam – vložení a odebrání prvku na pozici

Kroky vložení do seznamu l o velikost n na pozici k:

1. Kontrola velikosti pole: pokud $n + 1 > 2^k$ ($k = \lceil log_2(n) \rceil$), zvětšíme.
2. Posun všech prvků: $l[i + 1] = l[i]$ pro $i = n - 2 \ldots k$.
3. Přepsání prvku na pozici k.

Kroky odebrání prvku ze seznamu l o velikost n z pozice k:

1. Posun všech prvků: $l[i] = l[i + 1]$ pro $i = k \ldots n - 2$.
2. Kontrola velikosti pole: pokud $n - 1 < 2^{k-1}$, zmenšíme.

Seznam – vložení a odebrání prvku na pozici

Kroky vložení do seznamu l o velikost n na pozici k:

1. Kontrola velikosti pole: pokud $n + 1 > 2^k$ ($k = \lceil log_2(n) \rceil$), zvětšíme.
2. Posun všech prvků: $l[i + 1] = l[i]$ pro $i = n - 2 \ldots k$.
3. Přepsání prvku na pozici k.

Kroky odebrání prvku ze seznamu l o velikost n z pozice k:

1. Posun všech prvků: $l[i] = l[i + 1]$ pro $i = k \ldots n - 2$.
2. Kontrola velikosti pole: pokud $n - 1 < 2^{k-1}$, zmenšíme.

Asymptotická složitost

Složitost vložení prvku na pozici: $\Theta(n)$

Zásobník implementovaný polem

Uspořádaná, měnitelná kolekce prvků s duplicitami.

Podporované operace:

- přidej prvek na konec,
- vyber prvek na konci,
- odeber prvek z konce.

Zásobník implementovaný polem

Uspořádaná, měnitelná kolekce prvků s duplicitami.

Podporované operace:

- přidej prvek na konec,
- vyber prvek na konci,
- odeber prvek z konce.

Implementace:

- obdobné jako Seznam,
- tím, že přidáváme / odebíráme na / z konce, nemusíme prvky posouvat,
- amortizovaná složitost: $\mathit{Theta}(1)$.

Fronta implementovaná polem

Uspořádaná, měnitelná kolekce prvků s duplicitami.
Podporované operace:

- přidání prvku na konec,
- vybrání prvku na začátku,
- odebrání prvku ze začátku.

Implementace:

- používáme cyklické pole,
- detaily na KIV/IDT,
- amortizovaná složitost: $\mathit{Theta}(1)$.

Složitosti operací

Operace	Vysvětlení	Složitost
list[index]	Přístup k prvku na konkrétní pozici	O(1)
list.append(x)	Přidání prvku na konec seznamu	O(1)
list.pop()	Odebrání a vrácení posledního prvku seznamu	O(1)
list.pop(i)	Odebrání a vrácení prvku na i-té pozici	O(n)
list.insert(i, x)	Vložení prvku na konkrétní pozici	O(n)
del list[i]	Odebrání prvku na konkrétní pozici	O(n)
for x in list	Projití seznamu	O(n)
len(list)	Získání délky seznamu	O(1)
max(list)	Získání největšího prvku seznamu	O(n)
min(list)	Získání nejmenšího prvku seznamu	O(n)
x in list	Zkontrolování, zda je prvek v seznamu	O(n)
list1 + list2	Sloučení dvou seznamů	O(k)
list * k	Opakování seznamu k-krát	O(nk)
list.remove(x)	Odebrání prvního výskytu prvku	O(n)

Spojový seznam

Problém implementací polem

- Přidání prvku je někdy rychlé a někdy pomalé,
- občas potřebujeme záruku, že operace proběhne v nějakém "krátkém" čase,
- pole může zabírat místo v paměti, které aktuálně není potřeba.

Problém implementací polem

- Přidání prvku je někdy rychlé a někdy pomalé,
- občas potřebujeme záruku, že operace proběhne v nějakém "krátkém" čase,
- pole může zabírat místo v paměti, které aktuálně není potřeba.

Řešení

Spojový seznam

Spojový seznam

- chceme alokovat velké množství malých kousků paměti
- nemůžeme mít pro každý kousek vlastní identifikátor v programu

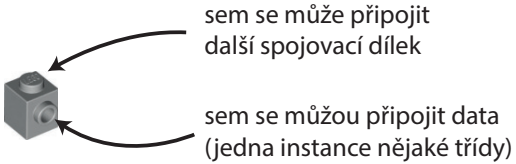
Spojový seznam

- chceme alokovat velké množství malých kousků paměti
- nemůžeme mít pro každý kousek vlastní identifikátor v programu

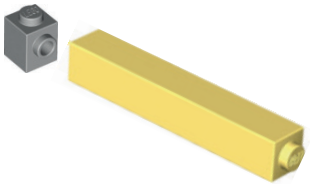
Řešení

- vyrobíme si spojovací prvek
- funguje trochu jako lego
- připojuje se na něj jeden kousek dat
- může se na něj připojit další spojovací prvek

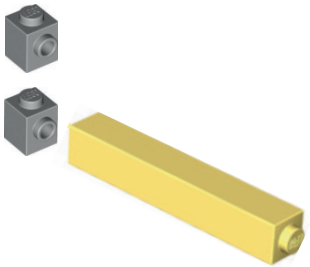
Spojový seznam



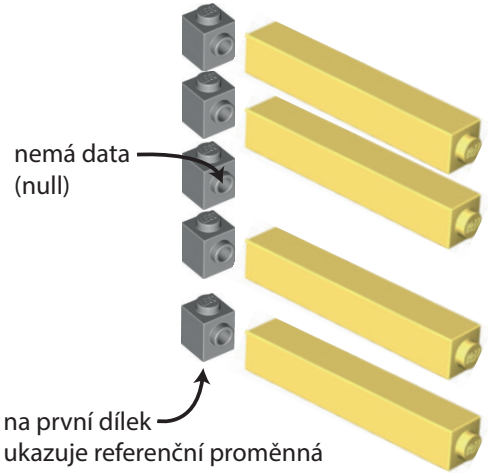
Spojový seznam



Spojový seznam



Spojový seznam



Spojový seznam v Pythonu

Třída s dvěma atributy:

- Data.
- Odkaz na další prvek.

Spojový seznam v Pythonu – kód

```
1  class Node:
2      def __init__(self, data):
3          self.data = data # Data uzlu
4          self.next = None # Ukazatel na další uzel

5  # Vytvoříme několik uzlů s různými daty
6  node1 = Node("A")
7  node2 = Node("B")
8  node1.next = node2
9  node3 = Node("C")
10 node2.next = node3

11 # Vypíšeme data prvních dvou uzlů
12 print(node1.data, node1.next.data)
```

Spojový seznam v Pythonu – Třída

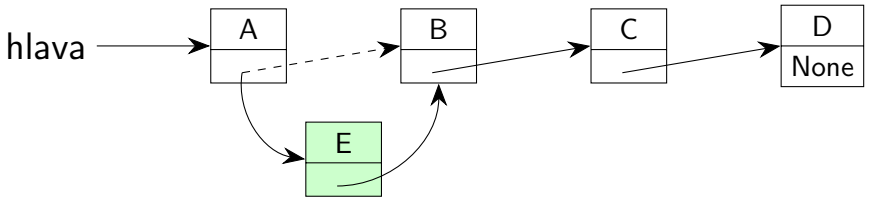
```
1  # Definujeme třídu pro spojový seznam
2  class LinkedList:
3    # Konstruktor pro nastavení hlavy seznamu
4    def __init__(self):
5      self.head = None # Hlava seznamu

6    # Vložení nového uzlu na začátek seznamu
7    def insert_at_head(self, data):
8      new_node = Node(data) # Vytvoříme nový uzel s daty
9      new_node.next = self.head # Nastavíme jeho další ukazatel na
       ↪  současnou hlavu seznamu
10     self.head = new_node # Nastavíme hlavu seznamu na nový uzel
```

Spojový seznam přidání prvku

Postup:

- Ukazatel nového prvku na následují prvek.
- Ukazatel aktuálního prvku nastavíme na nový prvek.



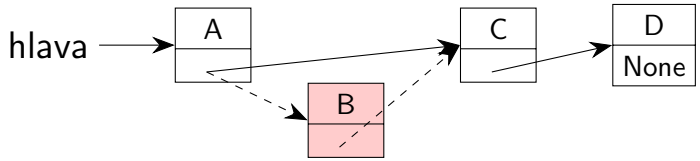
Spojový seznam přidání prvku – kód

```
1  class LinkedList:
2      ...
3      # Vložení nového uzlu za daný uzel v seznamu
4      def insert_after_node(self, data: int, node: Node) -> None:
5          new_node = Node(data) # Vytvoříme nový uzel s daty
6          new_node.next = node.next # Nastavíme ukazatel nového uzlu na
           ↪  další uzel daného uzlu
7          node.next = new_node # Nastavíme ukazatel daného uzlu na nový
           ↪  uzel
```

Spojový seznam odebrání prvku

Postup:

- Předchozímu prvku nastavíme ukazatel na následující prvek po mazaném prvku.




```
1  ...
2      # Smazání uzlu za daným uzlem v seznamu
3      def delete_after_node(self, node: Node) -> None:
4          node.next = node.next.next # Nastavíme další ukazatel daného uzlu
           ↪  na další uzel dalšího uzlu
```

Zásobník implementovaný spojovým seznamem

Uspořádaná, měnitelná kolekce prvků s duplicitami.

Podporované operace:

- přidej prvek na konec,
- vyber prvek na konci,
- odeber prvek z konce.

Zásobník implementovaný spojovým seznamem

Uspořádaná, měnitelná kolekce prvků s duplicitami.

Podporované operace:

- přidej prvek na konec,
- vyber prvek na konci,
- odeber prvek z konce.

Implementace:

- strukturu si virtuálně otočíme a konec si dáme na začátek,
- prvky přidáváme a odebíráme ze začátku (měníme ukazatel hlavy),
- hlídáme si jen konec zásobníku,
- složitost: $\Theta(1)$.

Fronta implementovaná spojovým seznamem

Uspořádaná, měnitelná kolekce prvků s duplicitami.
Podporované operace:

- přidání prvku na konec,
- vybrání prvku na začátku,
- odebrání prvku ze začátku.

Fronta implementovaná spojovým seznamem

Uspořádaná, měnitelná kolekce prvků s duplicitami.
Podporované operace:

- přidání prvku na konec,
- vybrání prvku na začátku,
- odebrání prvku ze začátku.

Implementace:

- strukturu si virtuálně otočíme a konec si dáme na začátek,
- prvky přidáváme na konec a odebíráme ze začátku (měníme ukazatel hlavy),
- hlídáme si jen konec fronty,
- složitost: $\Theta(1)$.

Srovnání implementací

Srovnání implementace polem VS spojovým seznamem

Mezi použitím implementací polem a spojovým seznamem je řada nuancí, které lépe pochopíte v předmětech KIV/IDT, KIV/PC a KIV/ZEP.

Srovnání implementace polem VS spojovým seznamem

Mezi použitím implementací polem a spojovým seznamem je řada nuancí, které lépe pochopíte v předmětech KIV/IDT, KIV/PC a KIV/ZEP.

Fundamentální rozdíly:

- Implementace polem umožňují vybrání prvků na pozici s konstantní složitostí.
- Implementace spojovým seznamem umožňují vložení a odebrání prvku na aktuální pozici s konstantní složitostí.
- Spojové seznamy je možné spojit s konstantní složitostí.
- Implementace polem nezatěžuje tolik systém správy paměti.

Srovnání implementace polem VS spojovým seznamem

Mezi použitím implementací polem a spojovým seznamem je řada nuancí, které lépe pochopíte v předmětech KIV/IDT, KIV/PC a KIV/ZEP.

Fundamentální rozdíly:

- Implementace polem umožňují vybrání prvků na pozici s konstantní složitostí.
- Implementace spojovým seznamem umožňují vložení a odebrání prvku na aktuální pozici s konstantní složitostí.
- Spojové seznamy je možné spojit s konstantní složitostí.
- Implementace polem nezatěžuje tolik systém správy paměti.

Datové struktury založené na spojových seznamech mohou být velmi dynamické. Jejich aplikaci uvidíme v dalších strukturách.

Kvíz II

Simulace systémů hromadné obsluhy

Systém hromadné obsluhy

Definice:

- Systém hromadné obsluhy (SHO) umožňuje efektivně modelovat a analyzovat opakující se operace a události.
- Simulace SHO nabízí flexibilní nástroj pro pozorování dopadů změn parametrů na systém.
- Základem jsou fronty, aktivní prvky simulující reálné objekty a princip modelování času.

Systém hromadné obsluhy

Definice:

- Systém hromadné obsluhy (SHO) umožňuje efektivně modelovat a analyzovat opakující se operace a události.
- Simulace SHO nabízí flexibilní nástroj pro pozorování dopadů změn parametrů na systém.
- Základem jsou fronty, aktivní prvky simulující reálné objekty a princip modelování času.

Uplatnění:

- Model SHO má široké praktické využití a je snadno implementovatelný.
- Oblasti, kde je potřeba efektivně řídit tok dat, zboží, osob či jakýchkoli jiných entit prostřednictvím procesů či stanic, kde může docházet k jejich zpracování, skladování nebo přepravě.
- Příklad: logistické systémy, výrobní linky, informační technologie (správa front požadavků v síťových zařízeních), zdravotnictví (řízení front pacientů) a mnoho dalších.

Modelování Času

Posouvání času:

- Diskrétní časové kroky, ve kterých aktualizujeme stav systému.
- Funkce, metoda (`tick()`) nebo cyklus posouvá čas o konstantní krok.

Modelování Času

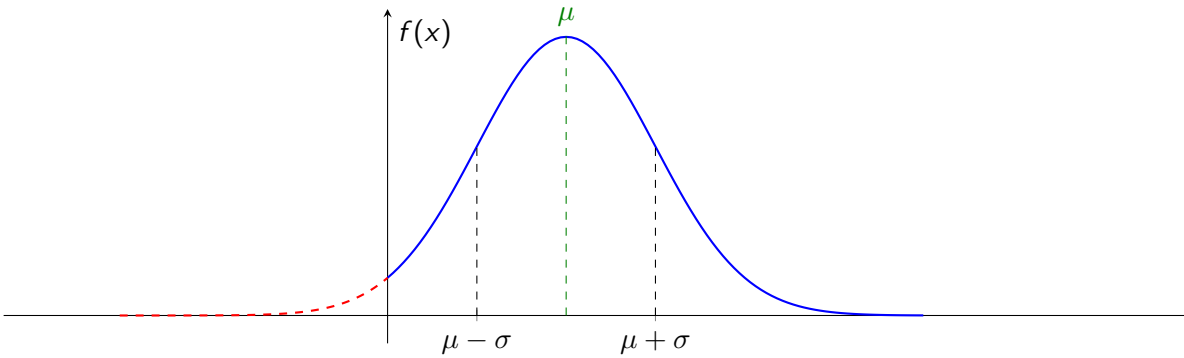
Posouvání času:

- Diskrétní časové kroky, ve kterých aktualizujeme stav systému.
- Funkce, metoda (`tick()`) nebo cyklus posouvá čas o konstantní krok.

Doba trvání akcí:

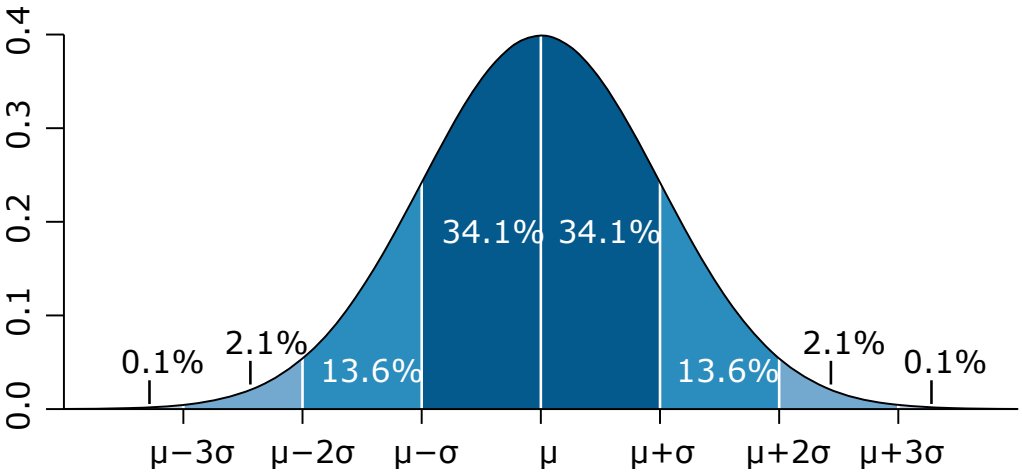
- Generování doby trvání a výskytu událostí na základě stanovených parametrů.
- Využíváme náhodnou proměnnou z exponenciálního, Poissonovo, normálního nebo jiného rozdělení pravděpodobnosti.

Normální rozdělení



POZOR na záporné hodnoty u normálního rozdělení!

Normální rozdělení – rozptyl



Generátor náhodných čísel

Generátor náhodných čísel: generuje posloupnost čísel statistickými testy nerozlišitelné od náhodné.

- Klíčový prvek pro simulaci náhodných procesů a událostí.
- Sekvence:
  - Absolutně náhodné: hardwarové generátory.
  - Pseudo-náhodné: softwarové generátory (používáme).
- Generují sekvence z daného rozdělení pravděpodobnosti.
- Softwarové generátory mají počáteční stav (seed).
  - Stejný seed $\rightarrow$ stejná sekvence.
  - Nastavení náhodné inicializace (seed) např. na základě aktuálního času.

Generátor náhodných čísel – příklad

```
1 import random

2 random.seed(42)         # Nastavení počátečního stavu generátoru
3 random.random()         # Náhodné číslo z rovnoměrného rozdělení v rozsahu
  ↪ [0.0, 1.0)
4 random.randint(1, 10) # Náhodné celé číslo  z rovnoměrného rozdělení z
  ↪ rozsahu [1, 10]

5 random.gauss(0, 1)      # Náhodné číslo z normálního rozdělení s parametry
  ↪ mu=0, sigma=1
```

References I

[1] GeeksforGeeks. “Accounting Method in Amortized Analysis”. In: (2023). URL: https://www.geeksforgeeks.org/accounting-method-amortized-analysis/.

[2] GeeksforGeeks. “Introduction to Amortized Analysis”. In: (2023). URL: https://www.geeksforgeeks.org/introduction-to-amortized-analysis/.

[3] GeeksforGeeks. “Potential Method in Amortized Analysis”. In: (2023). URL: https://www.geeksforgeeks.org/potential-method-in-amortized-analysis/.

[4] Robert E Tarjan. “Amortized computational complexity”. In: *SIAM Journal on Algebraic Discrete Methods* 6.2 (1985), s. 306–318.

[5] 42 Wolfsburg Library Wiki. *Cormen et al.: Introduction to Algorithms, third edition (2009) — 42 Wolfsburg Library Wiki*. [Online; accessed 22-March-2023]. 2022. URL: http://library.42wolfsburg.de/index.php?title=Cormen_et_al.:_Introduction_to_Algorithms,_third_edition_(2009)&oldid=804.