Datové typy

IB111: Datové typy

Data a algoritmizace

- jaká data potřebuji pro vyřešení problému?
- jak budu data reprezentovat?
- jaké operaci s nimi potřebuji provádět?

- Navržení práce s daty je velice důležité

Datový typ

- Datový typ
  - rozsah hodnot, které může proměnná daného typu přijmout
  - a množina operací, které jsou pro tento datový typ dovoleny/definovány.

Abstraktní datový typ (ADT)

- specifikuje strukturu a **operace** nad ní na základě dokumentované funkcionality (případně i složitost)
- **nezávisí na konkrétní implementaci**

- Důležité principy:
  - Abstrakce
  - Oddělení specifikace
  - Implementace

Základní ADT

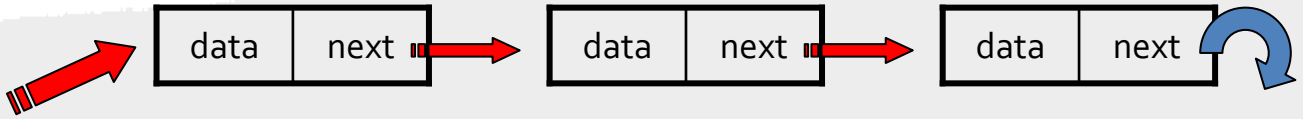
- Mezi základní ADT patří:
  - (lineární) seznam
  - zásobník
  - fronta
  - množina
  - slovník - asociativní pole
  - zobrazení
  - textový řetězec

(Zřetězený) seznam

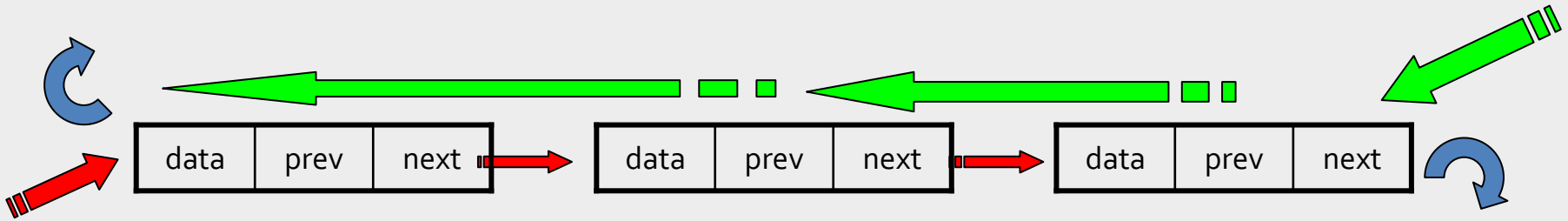
- Obsahuje posloupnost prvků
- Operace:
  - Přidání prvku
    - Na začátek
    - Na konec
    - Na určité místo (např. za/před určitý prvek)
  - Odebrání prvku
    - Ze začátku
    - Od konce
    - Konkrétní prvek (daný pozicí, hodnotou apod.)
  - Test prázdnosti seznamu

Typy (zřetězených) seznamů

- Jednosměrně zřetězený seznam

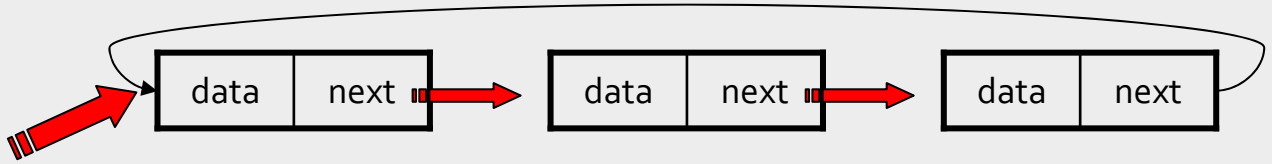


- Obousměrně zřetězený seznam



Typy (zřetězených) seznamů

- Kruhový seznam
  - Jednosměrný nebo obousměrný



Seznamy v Pythonu (opak.)

- Seznam prvků oddělený čárkami a uzavřený do hranatých závorek

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
```

- K prvkům můžeme přistupovat např. přes indexy
- Seznamy lze modifikovat

```
>>> a[0]
'spam'
>>> a[3]
1234
>>> a[-2]
100
>>> a[1:-1]
['eggs', 100]
>>> a[:2] + ['bacon', 2*2]
['spam', 'eggs', 'bacon', 4]
```

Seznamy v Pythonu (opak.)

- Na rozdíl od řetězců lze seznamy modifikovat

```
>>> a
['spam', 'eggs', 100, 1234]
>>> a[2] = a[2] + 23
>>> a
['spam', 'eggs', 123, 1234]
```

- Počet prvků – funkce len()

```
>>> a = ['a', 'b', 'c', 'd']
>>> len(a)
4
```

Seznamy: Užitečné funkce

list.append(*x*)	Přidá prvek x na konec seznamu
list.extend(*L*)	Přidá všechny prvky L na konec seznamu
list.insert(*i*, *x*)	Přidá prvek x před prvek na pozici i
list.remove(*x*)	Odstraní první prvek s hodnotou x
list.pop(i)	Odstraní prvek na pozici i
list.pop()	Odstraní poslední prvek
list.index(*x*)	Navrátí index prvního prvku s hodnotou x
list.count(*x*)	Navrátí počet výskytů hodnoty x v seznamu
list.sort()	Setřídí prvky seznamu
list.reverse()	Přehodí pořadí prvků (od posledního k prvnímu)
x in list	Test zda seznam obsahuje x

Seznamy: příklady

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print a.count(333), a.count(66.25), a.count('x')
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

Seznamy: příkaz del

- Příkaz del smaže prvek nebo jinou část seznamu

```
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]
>>> del a[0]
>>> a
[1, 66.25, 333, 333, 1234.5]
>>> del a[2:4]
>>> a
[1, 66.25, 1234.5]
>>> del a[:]
>>> a
[]
```

- Lze smazat i celou proměnnou

```
>>> del a
```

Vnořené seznamy

- Lze použít pro vícerozměrná data

```
>>> mat = [
...         [1, 2, 3],
...         [4, 5, 6],
...         [7, 8, 9],
...       ]
```

```
for i in [0, 1, 2]:
    for row in mat:
        print row[i],
    print
```

Vnořené seznamy - matice

- Vnořené seznamy
  - Vícerozměrná pole
  - Vhodné pro například pro matice
- Příklad: vytvoření nulové matice zadaných rozměrů

```
def nulova_matice(m,n):
    n=[[ 0 for row in range(n)] for col in range(m)]
    return n
```

Příklad: Matice

- Násobení matic

```
def vynasob(mat1,mat2):
    if(len(mat1[0]))!=len(mat2):
        return None
    else:
        n=nulova_matice(len(mat1),len(mat2[0]))
        for i in range(len(mat1)):
            for j in range(len(mat2[0])):
                for k in range(len(mat2)):
                    n[i][j]+=mat1[i][k]*mat2[k][j]
        return n
```

Příklad: (aritmetický) průměr

- Statistická veličina
  - Snaha zjistit „typickou hodnotu“
  - Nevýhoda: extrémní hodnoty ovlivňují průměr
- Výpočet:
  - součet všech hodnot dělený počtem prvků

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} X_i$$

```
def prumer(seznam):
    soucet = 0
    for prvek in seznam:
        soucet += prvek
    return float(soucet) / len(seznam)
```

Příklad: medián

- **Medián** je číslo, které rozdělí řadu podle velikosti seřazených čísel na dvě stejně početné poloviny.
- Nalezení mediánu:
  1. Seřadím čísla podle velikosti
  2. Vezmu číslo, která se nalézá uprostřed seznamu.
     - Pokud není nějaké číslo přesně uprostřed, tj. počet prvků je sudý, pak se obvykle za medián považuje aritmetický průměr hodnot na dvou místech u středu.

3, 5, 7, 12, 13, 14, 21, **23**, 23, 23, 23, 29, 39, 40, 56

3, 5, 7, 12, 13, 14, **21**, **23**, 23, 23, 23, 29, 40, 56

Příklad: medián

```
def median(seznam):
    seznam.sort()
    velikost=len(seznam)
    stred=velikost/2
    if(velikost%2==0):
        median = (seznam[stred]+seznam[stred-1]) / 2.0
    else:
        median = seznam[stred]
    return median

a=[1,4,3,9,4,58,26,4,4,56,4,5,2,7,1,8,7]
print "Median:",median(a)
print "Seznam:",a
```

```
>>>
Median: 4
Seznam: [1, 1, 2, 3, 4, 4, 4, 4, 4, 5, 7, 7, 8, 9, 26, 56, 58]
>>>
```

Zásobník (stack)

- Operace:
  - push (vložení)
  - pop (odstranění)
  - top (horní prvek)
  - empty (test prázdnosti)
- LIFO = Last In First Out
- Intuitivní příklad: sloupec knih
- Běžné použití:
  - procházení grafů
  - vyhodnocování výrazů
  - rekurze

Python: zásobník pomocí seznamu

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
[3, 4]
```

Příklad: postfixová notace

- Zápis matematických výrazů
  - Infixová notace
    - Zápis operátorů mezi operandy
    - Např. `1 + 2`
  - Prefixová notace
    - Zápis operátorů před operandy
    - Např. `+ 1 2`
  - Postfixová notace (reverzní polská notace)
    - Operátor následuje operandy
    - Není třeba používat závorky
    - Např. `1 2 +`

Příklad: postfixová notace

```
5 1 2 + 4 * + 3 -
```

Výraz je počítán zleva doprava.

Vstup	Operace	Zásobník	Popis
5	Ulož operand	5	
1	Ulož operand	5, 1	
2	Ulož operand	5, 1, 2	
+	Sečti	5, 3	Vyber dvě hodnoty (1, 2), zpracuj a ulož výsledek (3)
4	Ulož operand	5, 3, 4	
*	Vynásob	5, 12	Vyber dvě hodnoty (3, 4), zpracuj a ulož výsledek (12)
+	Sečti	17	Vyber dvě hodnoty (5, 12), zpracuj a ulož výsledek (17)
3	Ulož operand	17, 3	
-	Odečti	14	Vyber dvě hodnoty (17, 3), zpracuj a ulož výsledek (14)

Když je výpočet hotov, na vrcholu zásobníku je uložen výsledek, v tomto případě 14.

Příklad: postfixová notace

```
def vyhodnot(string):
    stack = []
    for token in string.split(" "):
        if set(token).issubset(set("0123456789.")):
            stack.append(float(token))
        elif token == '+':
            b = stack.pop()
            a = stack.pop()
            stack.append(a+b)
        elif token == '-':
            b = stack.pop()
            a = stack.pop()
            stack.append(a-b)
        else:
            print "Neznamy token: '%s'" % token
    return stack.pop()
```

Fronta (queue)

- operace:
  - push (vložení)
  - pop (odstranění)
  - top (první prvek)
  - empty (test prázdnosti)
- FIFO = First In First Out
- příklad:
  - zpracování příchozích požadavků serverem

Python a fronta

- lze pomocí seznamu, ale pomalé
  - Protože přidávání a odebírání ze začátku seznamu vyžaduje posun všech prvků…
- použití knihovny collections:

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")           # Terry arrives
>>> queue.append("Graham")          # Graham arrives
>>> queue.popleft()                 # The first to arrive now leaves
'Eric'
>>> queue.popleft()                 # The second to arrive now leaves
'John'
>>> queue                           # Remaining queue in order of arrival
deque(['Michael', 'Terry', 'Graham'])
```

Seznam x zásobník x fronta

- Zásobník a fronta jsou podobné seznamu
  - Ale poskytují méně možných operací
    - Např. přidání jen na začátek/konec
- Proč používáme struktury, které nám nabízejí méně funkcí?
- Pokud se omezíme na více specifický datový typ mohou být:
  - operace rychlejší a/nebo
  - paměťové nároky menší
- Při návrhu nám pomáhá uvědomit si, že potřebujeme např. zásobník a ne obecně seznam…

Množina (set)

- operace:
  - insert (vložení)
  - find (test na přítomnost)
  - delete (odstranění)
- příklady:
  - evidování navštívených vrcholů při prohledávání
  - počítání „průniku“ informací ze dvou souborů

Množiny v Pythonu

- Množina
  - Neuspořádaná kolekce dat bez duplicitních prvků

len(s)	počet prvků množiny
x in s	Test zda prvek x se nachází v množině s
s1 <= s2	Test zda s1 je podmnožinou s2
union nebo operátor |	Sjednocení množin
intersection nebo operátor &	Průnik množin
difference nebo operátor -	Rozdíl množin
symmetric_difference nebo operátor ^	Položky z jedné z množin nenacházející se v množině druhé

Množiny v Pythonu: příklad

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> fruit = set(basket)               # create a set without duplicates
>>> fruit
set(['orange', 'pear', 'apple', 'banana'])
>>> 'orange' in fruit                 # fast membership testing
True
>>> 'crabgrass' in fruit
False

>>> # Demonstrate set operations on unique letters from two words
...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                 # unique letters in a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b                             # letters in a but not in b
set(['r', 'd', 'b'])
>>> a | b                             # letters in either a or b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b                             # letters in both a and b
set(['a', 'c'])
>>> a ^ b                             # letters in a or b but not both
set(['r', 'd', 'b', 'm', 'z', 'l'])
```

N-tice (tuples) v Pythonu

- N-tice podobné seznamům
- Nejsou měnitelné (stejně jako řetězce)

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

```
>>> x, y, z = t
```

Slovník (dictionary, map)

- Slovník, dictionary, map, asociativní pole
  - zobecnění množiny, pole
- dvojice klíč, hodnota
  - klíče jsou unikátní
  - neuspořádané
- příklady:
  - počet výskytů jednotlivých slov v textu
  - „kešování“ výsledků časově náročných výpočtů

Slovník v Pythonu

```
>>> tel = {'jack': 4098, 'sape': 4139}
```

- {}
- Klíč a hodnotu oddělujeme dvojtečkou
  - U jednoduchých klíčů lze použít i =

```
>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

- Slovník lze vyrobit i n-tic (tuples)
- Slovník je modifikovatelný

Slovník v Pythonu: příklad

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> tel.keys()
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
```

Výpis slovníku

- Jednotlivé položky slovníku můžeme iterovat pomocí „.iteritems()“

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.iteritems():
...     print k, v
...
gallahad the pure
robin the brave
```

Příklad: frekvence slov

- Analyzujeme předložený text
- Pro každé slovo vypíšeme kolikrát se v textu vyskytuje
- Výsledek lze využít jako metriku podobnosti dokumentů apod.

```
Seznam nejcasteji pouzitych slov:
the         27801
and         26834
i           20296
to          19749
of          18299
a           14620
you         13713
my          12473
that        11149
in          11060
is           9620
not          8753
for          8268
with         8049
me           7771
it           7710
be           7106
your         6882
this         6879
his          6853
```

Příklad: frekvence slov - výpočet

```
def frekvence_slov(soubor,n):
    text = open(soubor,'r').read()
    text = string.lower(text)
    for ch in '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~':
        text = string.replace(text, ch, ' ')
    seznam_slov = string.split(text)
    frekvence = {}
    for slovo in seznam_slov:
        if frekvence.has_key(slovo):
            frekvence[slovo] += 1
        else:
            frekvence[slovo] = 1
```

Příklad: frekvence slov - výpis

```
def srovnej((s1,p1), (s2,p2)):
    if p1 > p2:
        return -1
    elif p1 == p2:
        return cmp(s1, s2)
    else:
        return +1
```

```
vysledek = frekvence.items()
vysledek.sort(srovnej)
print "Seznam nejcasteji pouzitych slov:"
for i in range(n):
    s, p = vysledek[i]
    print s,"\t",p
```

Příklad: morseovka

```
morse = {'A':'.-', 'B': '-...', 'C': '-.-.',
         'D': '-..' } # atd

def prevod_morse(s):
    vystup = ''
    for i in range(len(s)):
        if morse.has_key(s[i]):
            vystup += morse[s[i]] + '|'
    return vystup
```

Příklad: převodník

```
sub = {
    "morse": [".-","-...","-.-.","-..", # atd ],
    "braille": ["BWWWWW","BWBWWW","BBWWWW", # atd ],
}

def prevod(s, kodovani):
    vystup = ''
    if not sub.has_key(kodovani): return None
    for i in range(len(s)):
        if ord('A') <= ord(s[i]) <= ord('Z'):
            c = ord(s[i])-ord('A')
            vystup += sub[kodovani][c] + '|'
    return vystup
```