Datové struktury.

Datové typy. Základní a odvozené datové struktury. Hashování*.

Tomáš Bayer | bayertom@natur.cuni.cz

Katedra aplikované geoinformatiky a kartografie, Přírodovědecká fakulta UK.

Obsah přednášky

1. Základní datové typy

Logické, celočíselné, reálné a znakové.

Existují prakticky ve všech programovacích jazycích.

Definován rozsah hodnot v bajtech (velikost uloženého čísla).

Python dynamicky typovaný jazyk, datový typ u proměnné nemusíme uvádět.

Avšak uvnitř silná typová kontrola.

Celočíselné datové typy:

Reprezentace diskrétních jevů, výpočty "bezchybné": `int`, `long`.

Velikost shora neomezená (není běžné u jiných programovacích jazyků).

Logické datové typy:

Reprezentace stavů pravda/nepravda: `Boolean`

Reálné datové typy:

Reprezentace spojitých jevů: `float`, `double`.

Velikost shora/zdola omezená, výpočty zatíženy chybou.

Standardizace IEEE 754.

Znakové typy:

Vyjádření textových informací, znak: `char`, slovo: `string`.

Znak lze reprezentovat celým číslem.

ASCII nebo UNICODE sady.

2. Práce s literály

Literál: přímý zápis hodnoty učitého typu v programovacím jazyce.

Celé číslo :

```
123456789
```

Reálné číslo: pevná/plovoucí řádová čárka, stejná absolutní/relativní přesnost

```
3.14159        1.0e-15          #FX vs FP
```

Malá čísla méně přesná, problematické při některých aritmetických operacích.

Nečíselný výsledek, NaN (Not a Number):

Výsledek operace není definován (např. odmocnina ze záporného čísla).

```
math.sqrt(-1)
>>> ValueError: math domain error
```

Nekonečno, Inf (Infinity):

Výsledek aritmetických operací (např. dělení 0).

```
math.sqrt(1/0)
>>> ZeroDivisionError: division by zero
```

Logická hodnota:

```
True        False
```

Řetězce: single, double, tripple quoted

```
'Hello'       "Hello"
"""We are ready to       #Viceradkovy text
     learn "Python""")
```

3. IEEE 754 (1985)

Definice standardů pro aritmetiku s reálnými čísly (plovoucí řádová čárka).

Implementováno v programovacích jazycích.

Nejdůležitější body:

- *Přesnost*
  Jednoduchá přesnost (32 bit), dvojitá (64 bit), dvojitá rozšířená (80 bit).

Typ	Platné cifry	Reprezentace	Chyba
`float`	7	32 bit	$2.38 \cdot 10^{-7}$
`double`	15	64 bit	$1.42 \cdot 10^{-14}$

- *Nečíselný výsledek, NaN (Not a Number).*
  Výsledek operace není definován (např. odmocnina ze záporného čísla).

```
math.sqrt(-1)
>>> ValueError: math domain error
```

- *Nekonečno, Inf (Infinity)*
  Výsledek aritmetických operací (např. dělení 0).

```
math.sqrt(1/0)
>>> ZeroDivisionError: division by zero
```

- *Přetečení, podtečení*
  Mezní hodnoty: $x_{min} = 2^{-126}$, $x_{max} = 2^{127}$.

  $x < 0 \wedge x < -x_{max}$: Negative Overflow,
  $x < 0 \wedge x > -x_{min}$: Negative Underflow,
  $x > 0 \wedge x < x_{min}$: Positive Underflow,
  $x > 0 \wedge x > x_{max}$: Positive Overflow.

4. Problémy při práci s reálnými čísly

Problém 1: Zaokrouhlení při matematikých operacích

V podmínkách místo a==b používat abs(a-b)<eps

```
acos(cos(1))==1
>>> False
```

Problém 2: Odečtení malého čísla od čísla

Výsledek operace je špatný.

```
x = 1.0
y = 0.00000000000000000005
z = x - y
>>> 1.0
```

Problém 3: Přičtení malého čísla k velkému číslu

Porovnání dá špatný výsledek.

```
x = 1.0e20
x == x + 1
>>> True
```

Problém 4: Asociativita pro malá a velká čísla

Zdánlivá nefunkčnost asociativity.

```
a = 1
b = 1e20
c = -1e20
a + ( b + c )
( a +  b ) + c

>>> 1.0
>>> 0.0
```

5. Kódování znaků

1 znak: char (samohláska, souhláska, číslo, speciální znak).

Ze znaků skládány posloupnosti, tzv. *textové řetězce,* odpovídají slovům.

Interní reprezentace znaků v PC celočíselná.

Standardizace znakových sad v informatice:

- *ASCII (American Standard Code for Information Interchange), 1967*
  127 znaků (písmena, číslice, spec. znaky), 7b + 1b.
  Nevyhovující, reprezentace malého množství znaků.
  Problémy s ČJ, celkem 6 speciálních kódování: CP 1250, ISO 8859-2, Latin 2...
- *UNICODE (Universal Coded Character Set), 1991*
  16b kódování, nástupce ASCII, 1 114 112 znaků.
  Umožňuje vyjádřit libovolný znak z libovolného jazyka.
  Nejčastější varianta kódování: UTF-8.

Speciální (řídící) znaky reprezentovány ***Escape sekvencemi.***

Uvozující znak \ (backslash).

Escape sekvence	UNICODE	Význam
`\n`	`\u000A`	Nová řádka
`\r`	`\u000D`	Návrat na začátek řádku
`\t`	`\u0009`	Tabulátor
`\\`	`\u005C`	Zpětné lomítko
`\'`	`\u002C`	Apostrof
`\"`	`\u0022`	Uvozovky

```
print("We are \t ready to \n learn \"Python\"")
>>> We are   ready to
>>> learn "Python"
```

6. ASCII tabulka

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

7. Datové struktury

Data reprezentována různými datovými typy.

Uchovávána v datových strukturách.

Pro efektivní práci s daty nutné zvolit:

- odpovídající datový typ,
- vhodnou datovou strukturu.

Dělení datových struktur:

Datové struktury děleny do dvou kategorií:

- *Základní datové struktury:*
  Vyskytují se téměř ve všech programovacích jazycích.
  Za běhu programu nemění svůj rozsah.
- *Odvozené datové struktury:*
  Nazývány jako *abstraktní datové struktury.*
  Často implementovány jako objekty.
  Za běhu programu mohou měnit svůj rozsah.

Volba optimální datové struktury:

Nutno zohlednit mnoho faktorů, kritický vliv na efektivitu běhu SW.

Velikost dat, požadovaná funkcionalita, typické operace, průběžná změna dat.

Optimalizace na úrovni datových struktur.

8. Dělení datových struktur

Dělení do dvou skupin:

- *Základní datové struktury:*
  - Proměnná (`Variable`).
  - Pole (`Array`).
  - Struktura (`Structure`).
  - Objekt (`Object`).
- *Odvozené datové struktury:*
  - Seznam (`List`).
  - Strom (`Tree`).
  - Zásobník (`Stack`).
  - Fronta (`Queue`).
  - Prioritní fronta (`Priority Queue`).
  - Množina (`Set`).
  - Tabulka (`Table`).

9. Proměnná

Pojmenované místo v paměti počítače.

Je v ní uložena hodnota určitého datového typu.

Typ proměnné:

Specifikace typu proměnné ovlivňuje:

A) Určuje množinu hodnot, kterých proměnná může nabývat.

B) Množství paměti potřebné pro její uložení.

C) Operace, které lze s proměnnou provádět.

Před použitím se provádí deklarace a inicializace (lze spojit).

Deklarace proměnné (staticky typované jazyky):

V deklaraci uveden *datový typ* proměnné a její *identifikátor*.

```
int i; double k;              //Deklarace
```

Inicializace proměnné:

Přiřazení výchozí hodnoty.

Nepoužívat neinicializované proměnné!

```
i = 7; k = 3.0;               //Inicializace
int i = 7; double k = 3.0;    //Deklarace+inicializace
```

Dynamicky typované jazyky:

Deklarace se nepoužívá, proměnná se pouze inicializuje.

```
i = 7       #Na první pohled není jasný typ
```

Type Hints:

Možné uložit informaci o typu proměnné, není vynutitelné.

Ověřování při statické typové kontrole.

```
i : int = 7       #Type hint, zvyseni prehlednosti
```

10. Druhy proměnných

Lokální proměnné:

Deklarovány ve funkcí či procedurách.

```
def g(x):
    y = math.sin(x)     #Lokalni promenna
    return y
```

Nazývány jako automatické proměnné, alokovány v zásobníku (Stack).

Existují pouze po dobu běhu procedury/funkce.

Úspora paměti, nepotřebné proměnné nezabírají místo.

Globální proměnné:

Vytvořeny při spuštění programu, zanikají po ukončení programu.

Přidělování paměti realizováno již v době překladu (kompilátor).

Statická alokace (V CPythonu neplatí, vše je objektem!).

```
def g(x):
    global y            #y je nyni globalni
    y = math.sin(x)
    return y
a = 1.0                 #Mimo telo funkce, globalni
```

Dynamické proměnné:

Vznikají za běhu programu.

Alokovány na haldě (Heap), přidělování paměti řídí OS (ne kompilátor).

Mohou zanikat automaticky nebo manuálně (příkazem).

Standardní přístup v Pythonu: konejnery, objekty,...

```
Point p(10, 10)
```

11. Přiřazovací příkaz

Přiřazuje proměnné hodnotu odpovídajícího datového typu.

Dochází k inicializaci hodnoty proměnné.

```
a = 5                              #Cele cislo
b = 0.00015                        #Realne cislo, FX
c = 1.5e-4                         #Realne cislo, FP
a, b, c = 5, 0.00015, 1.5e-4       #Kombinace
a : int = 5                        #Type hint
```

Konstanta: hodnota se nemění, zpravidla velkými písmeny

```
PI = 3.141592657
```

Obecný tvar

```
promenna = vyraz                   #Prirazeni hodnoty vyrazu
```

Výraz: kombinace proměnných, konstant, funkcí, operátorů (artimetických, logických).

Proměnné vystupují:

- v aritmetických operacích: výrazy

```
dx = xb - xa
dy = yb - ya
dist = (dx * dx + dy * dy)**0.5;
```

- v logických operacích: podmínky

```
if eps > 0
```

- předávány jako parametry funkcí

```
distance (xa, ya, xb, ya);
```

12. Typové kontroly

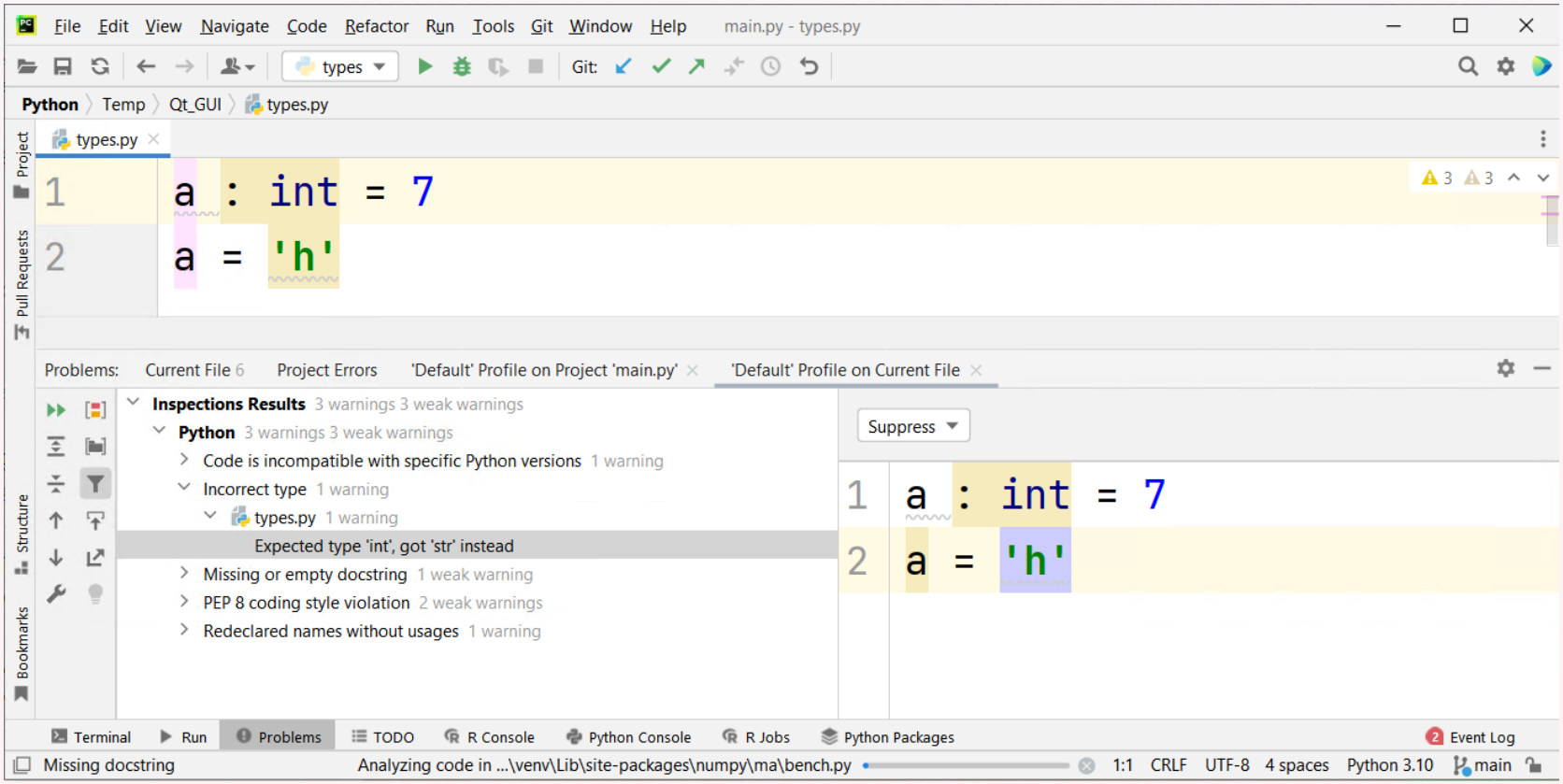
Type Hints umožňují využit typovou kontrolu:

```
a : int = 7                          #Promenna a je typu int
b = 'h'                              #Nyni priradime retezec
```

Při kompilaci nevznikne chyba, jedná se pouze o typovou anotaci.

Není vynutitelná, Python má dynamické typování.

Nalezení chyby: Static/Dynamic Code Analysis.



13. Přetypování

Při přiřazování dochází ke konverzím mezi různými datovými typy.

Přiřazovaná hodnota (vpravo) konvertována na typ proměnné jíž přiřazujeme (vlevo):

```
variable = value        nebo    variable2 = variable1
(type 2)   (type 1)             (type 2)    (type 1)
```

Výsledkem změna datového typu proměnné (type 1-> type 2).

Varianty přetypování:

- Implicitní (automatické).
- Explicitní (vynucené).

Zjištění typu proměnné v Pythonu:

příkaz `type(variable)`

```
a = 12
type(a)
>>> <class 'float'>
```

14. Implicitní a explicitní přetypování

Implicitní přetypování:

Přetypování proměnné s nižším rozsahem na proměnnou s vyšším rozsahem.

Probíhá automaticky, nedochází při ní ke ztrátě informace.

Pořadí (Python): boolean->int->float->(complex)->string.

```
a = 1
b = 9.0
c = a + b; #implicitni konverze na float
type(c)
>>> <class 'float'>
```

Explicitní přetypování:

Přetypování proměnné s vyšším rozsahem na proměnnou s nižším rozsahem.

```
var2 = type(var1)
```

Nutno vynutit uvedením cílového datového typu.

Pozor: dochází ke ztrátě informace!

Pořadí (Python): string->(complex)->float->int->boolean.

```
d = int(c) #explicitni konverze na int
type(d)
>>> <class 'int'>
```

Konverzní funkce (Python):

```
int(x), long(x), float(x), str(x), chr(x), ord(x)
```

15. Aritmetické operátory

Slouží k realizaci základních aritmetických operacích.

Figurují v aritmetických výrazech.

Vyhodnocování výrazu z leva do prava dle priority:

1. operace násobení/dělení/celočíselné dělení,
2. poté zbývající operace.

Změna priority vyhodnocování s použitím závorek.

Počet pravých a levých závorek by měl být stejný.

```
a = 3 + 3 * 3 #a=12
a = (3 + 3) * 3 #a=18
```

Přehled základních aritmetických operátorů:

+	Sčítání
-	Odečítání
*	Násobení
/	Dělení
//	Celočíselné dělení
**	Mocnina
%	Zbytek po celočíselném dělení.

16. Operátory přiřazení

Zkrácený zápis běžných aritmetických operací.

Umožňuje efektivnější zápis aritmetických operací ve výrazech.

Přehled operátorů přiřazení:

a=5	
a+=5	a=a+5
a-=5	a=a-5
a*=5	a=a*5
a/=5	a=a/5
a%=5	a=a%5
a**=5	a=a**5
a//=5	a=a//5

Používat rozumně, jinak nepřehledný zápis

```
b += a * n -= 1
```

Standardní zápis

```
n = n - 1
b = b + a * n
```

17. Relační operátory

Použití při konstrukci logických podmínek: příkaz pro větvení, cykly, výrazy.

Vyhodnocování podmínky z leva do prava dle priority:

1. negace,
2. konjunkce,
3. ostatní.

Změna priority vyhodnocování s použitím závorek.

Přehled základních relačních operátorů:

`==`	Rovná se	^	XOR
`!=`	Nerovná se	<	Menší
<>	Nerovná se	>	Větší
`&`	Logický součin	<=	Menší nebo rovno
|	Logický součet	>=	Větší nebo rovno
~	Negace		

Pozor na záměnu: `= vs. ==`

Porovnávání dvou reálných čísel:

```
a == b: #Nikdy nepouzivat
```

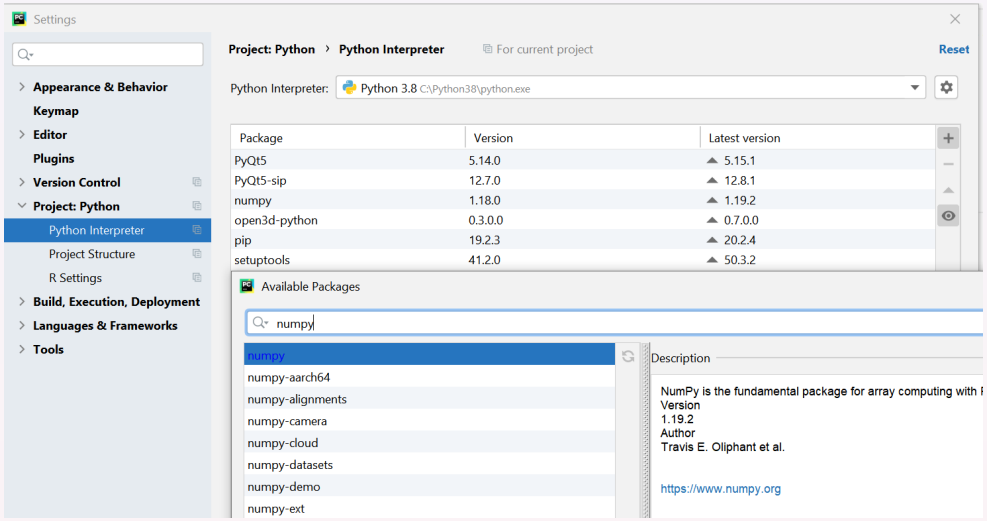
Testujeme hodnotu $|a - b|$ nebo $|(a - b)/a|$ vzhledem k $\varepsilon \gg 0$:

```
abs(a - b) < eps
```

18*. Matematika a Python

Python disponuje velkým množstvím knihoven: `NumPy`, `SciPy`

https://pip.pypa.io/en/stable/installing/



Instalace balíků `SciPy`:

py -m pip install –user numpy scipy matplotlib ipython jupyter pandas sympy nose

`SciPy`: Knihovna pro vědecké výpočty, vizualizace.

`NumPy`: rozsáhlý balík (lineární algebra / analýzy), “light” verze Matlabu.

Součást balíku `SciPy`.

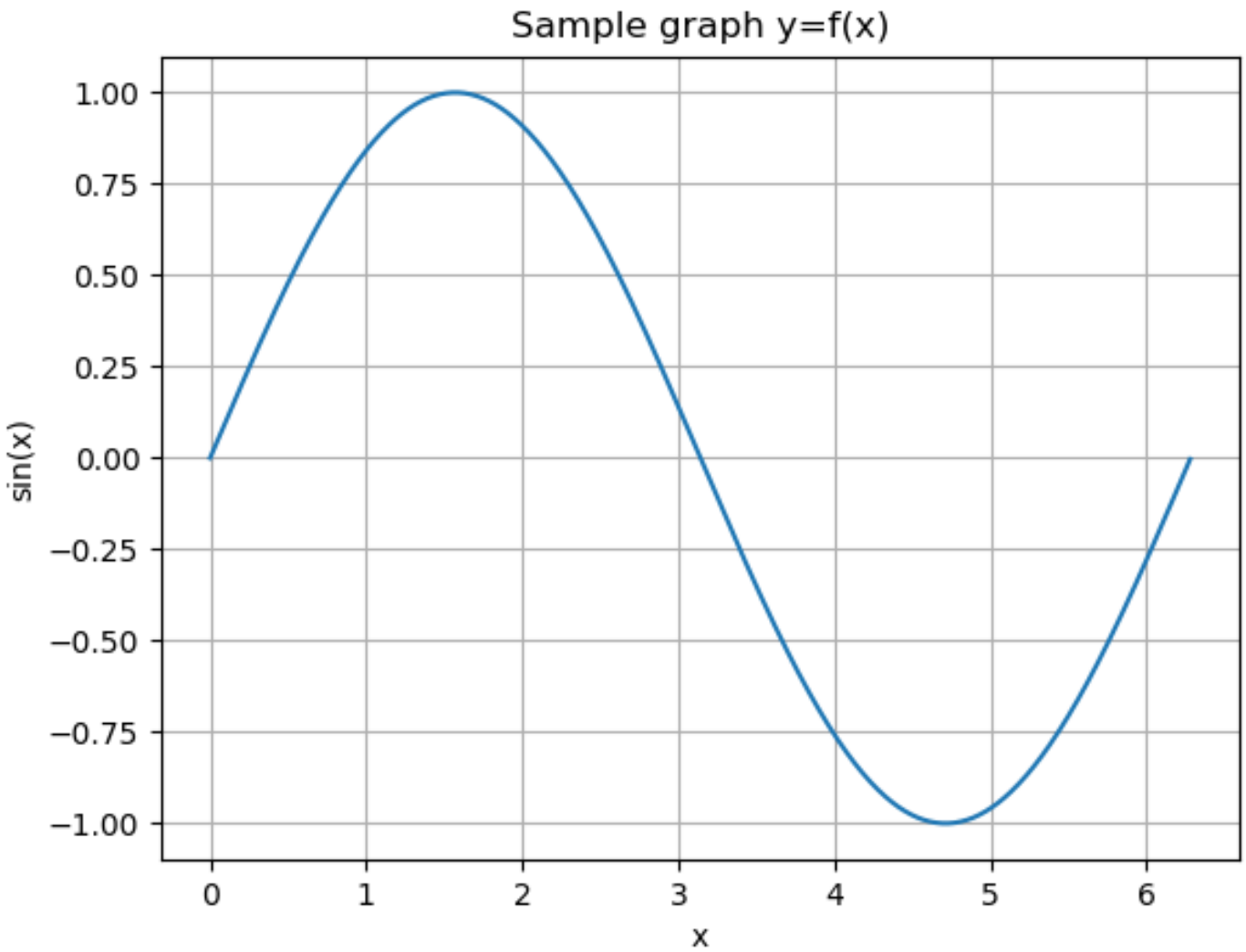
Ukázka pseudoinverze matice v `NumPy`:

```
import numpy as np
from numpy.linalg import pinv
A = np.matrix([[1., 2.], [2., 4.]])
AI = pinv(A)
print(AI)
>>> [[0.04 0.08]  [0.08 0.16]]
```

19*. Kresba grafu $y = \sin(x)$

```
import matplotlib
import matplotlib.pyplot as plt
from numpy import sin
from numpy import pi
from numpy import arrange

# Data for plotting
t = arange(0.0, 2.0*pi, 0.01)
s = sin(t)

#Plot
fig, ax = plt.subplots()
ax.plot(t, s)

#Labels
ax.set(xlabel='x', ylabel='sin(x)',
title='Sample graph y=f(x)')
ax.grid()

#Export
fig.savefig("test.png")
plt.show()
```

20*. Ukázka grafu



21. Seznam (List)

Datová struktura, tvoří ji uspořádaná posloupnost položek.

Položka $\equiv$ uzel (Node).

Každý uzel obsahuje odkaz na následující/předchozí uzel.

Rekurzivní struktura: odkazy na položky stejného typu.

Vlastnosti seznamu:

+ Rychlé přidání prvků na počátek/konec seznamu $O(1)$.
- Přidání prvku na jiné místo $O(N)$.
- Hledání prvku $O(N)$.
- Mazání prvku $O(N)$.

Většina operací funkcí počtu prvků N: zpomalování operací.

Vhodný pro procházení prvků “popořadě” $\Rightarrow$ sekvenční uspořádání dat.

Nevhodný pro přímý přístup k prvkům (dle implementace).

Typy seznamů:

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

22. Ukázky seznamů

Jednosměrný seznam:

Každý prvek odkazuje na předchozí/následující prvek seznamu.
První/poslední prvek seznamu odkazují na `None`.

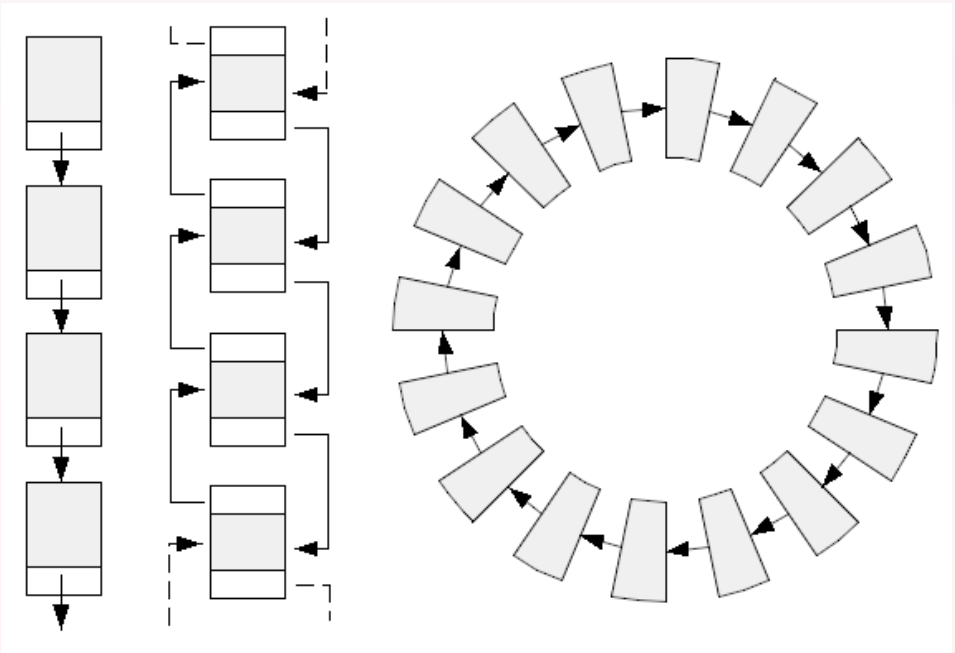
Obousměrný seznam:

Každý prvek odkazuje na předchozí/následující prvek seznamu.
První/poslední prvek seznamu odkazují na `None`.

Kruhový seznam:

Jednosměrný i obousměrný.

Poslední/první prvek seznamu odkazuje na první/poslední prvek seznamu.



23. Seznamy v Pythonu

Výchozím typem je obousměrný seznam *L*.

Každá položka může obsahovat objekty jiného typu (v praxi nepoužívat!)

Vytvoření seznamu:

```
L =[]                          #Prazdny seznam
L = [123, 456.3, -37.3]        #Seznam se 3 polozkami
```

Zpravidla ukládáme objekty stejného typu, lze kombinovat (nedoporučuje se)

```
L = [123, 456.3, ''Hello'', -37.3]
```

Seznamy mohou být i vnořené

```
L = [123, 456.3, ''Hello'', -37.3, ''World'', False, [-1, ``PC'', False]]
```

Přístup prostřednictvím obousměrného indexu, v hranatých závorkách.

L[-7]	L[-6]	L[-5]	L[-4]	L[-3]	L[-2]	L[-1]		
123	456.3	''Hello''	-37.3	''World''	False	-1	''PC''	False
L[0]	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]		

Platí:

```
L[0] == L[-7] = 123
L[2][1] == L[-5][-4] = ``e''
L[6] == L[-1] = [-1, ``PC'', False]]
L[6][2] == L[-1][2] = ``PC''
```

24. Základní operace se seznamem

Opakování přidávané položky:

```
L = [5]*7   #Seznam tvoren [5, 5, 5, 5, 5, 5 ,5]
```

Práce s částí seznamu, používány řezy (slices):

```
K = L[2:4]
```

Délka seznamu: příkaz `len()`

```
n = len(L)
```

Přidání na konec seznamu: metoda `append()`

```
L.append(“x”)
```

Přidání prvku na pozici *p*, následující posunuty o 1 vpravo: metoda `insert()`

```
L.insert(p,”x”)
```

Spojení 2 seznamů - operátor +:

```
L = [1, 2, 3] + [4, 5]
```

Nalezení položky na pozici *p* v seznamu: metoda `index()`

```
i = L.index(4)
```

Odstranění posledního prvku ze seznamu: metoda `pop()`

```
n = L.pop()
```

Odstranění položky na pozici *p* ze seznamu: metoda `del()`

```
L.del(3)
```

25. Operace s řetězci v Pythonu

Práce s řetězci podobná práci se seznamem, každý znak má index.

Index obousměrný.

s[-11]	s[-10]	s[-9]	s[-8]	s[-7]	s[-6]	s[-5]	s[-4]	s[-3]	s[-2]	s[-1]
H	e	l	l	o		w	o	r	l	d
s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]	s[10]

Lze vytvářet podřetězce (slices):

```
seq[index] #1 znak
seq[start:end] #interval od-do
seq[start:] #vsechny znaky od
seq[:end] #vsechny znaky od
seq[start:end:step] #Interval od-do s krokem
```

Příklady:

```
s[0:10:2]
>>> Hlwr
s[::2]
>>> Hlwl
s[::-2]
>>> drwolH
```

26. Ukázka operací s řetězci

Replikace řetězců:

```
print(s * 3)
>>> Hello worldHello worldHello world
```

Konverze znaku na číslenou reprezentaci:

```
ord(d)
>>> 100
```

Konverze číselné reprezentace na textovou:

```
chr(100)
>>> d
```

Znak s maximálním/minimálním ASCII kódem:

```
min(s)
>>>  #mezera
max(s)
>>> w
```

Delimitace:

```
s = "Hello*my*new*world"
s.split('*')
>>>  ['Hello', 'my', 'new', 'world']
```

Délka řetězce:

```
len(s)
      >>> 13
```

Výskyt podřetězce v řetězci:

```
"wo" in s
      >>> True
```

27*. Maticové operace

Použití vnořeného seznamu (2D seznam).

Každý prvek 1D seznamu tvořen dalším 1D seznamem.

```
A = [[1, 2, 3], [4, 5, 6], [1, 0, 1]]
```

Nepodporuje maticové operace, nutno implementovat samostatně.

V praxi se moc nepoužívá.

Standardem využití knihovny NumPy

https://numpy.org/

Rozsáhlá knihovna, stovky funkcí.

Ukázka řešení MNČ pseudoinverzí s využitím NumPy

$$h = (A^T A)^+ (A^T l)$$

```
import numpy
from numpy import *
A = matrix([[1., 2., 3.], [4., 5., 6.], [7., 8., 9.]])
B = matrix([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])
l = array([[1, 2, 3]]).
C = A + B
D = A * B
h = inv(transpose(A) * A) * transpose(A) * l #MNC
```

28. Zásobník

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

Pracovat lze pouze s prvkem, který je na vrcholu zásobníku.

Reprezentuje model LIFO (Last In - First Out), např. batoh.

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

Použití při sekvenčním zpracovávání dat.

Dno zásobníku:

Nejspodnější prvek v zásobníku, přidán jako první.

Vrchol zásobníku:

Nejvrchnější prvek v zásobníku, přidán jako poslední.

Python nemá specifický datový typ pro zásobník, použit List.

Přidání prvku do zásobníku:

Použití metody `append()`, přidání na konec seznamu.

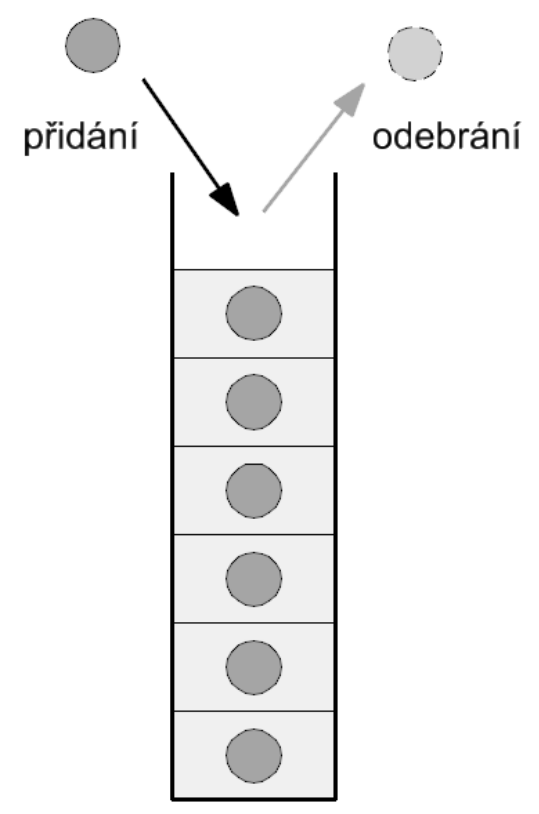
```
S = []
S.append(1)
S.append(2)
S.append(3)
```

Odstranění prvku ze zásobníku:

Použití metody `pop()`, odebíráme z konce seznamu.

```
S.pop()
```

29. Ukázka zásobníku



Použití: náhrada rekurze, vyhodnocování výrazů...

30. Fronta

Odebíráme data ve stejném pořadí, v jakém jsem je do ní uložili.

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

Reprezentuje model FIFO (First In-First Out).

Využití při sekvenčním zpracování dat.

Konec fronty:

Prvek na poslední pozici ve frontě, přidán jako poslední.

Čelo fronty:

Prvek na první pozici ve frontě, přidán jako první. V Pythonu implementace s využitím List (vytvoření, přidání, viz Stack).

Odebíráme první prvek seznamu:

```
S.pop(0)  #Jako Stack, odebirame ze zacatku.
```

Alternativní implementace s Queue

```
import queue
Q = queue.Queue #Volba typu fronty- bezna
```

Přidání prvku do fronty:

Použití metody `put()`, přidání na konec fronty.

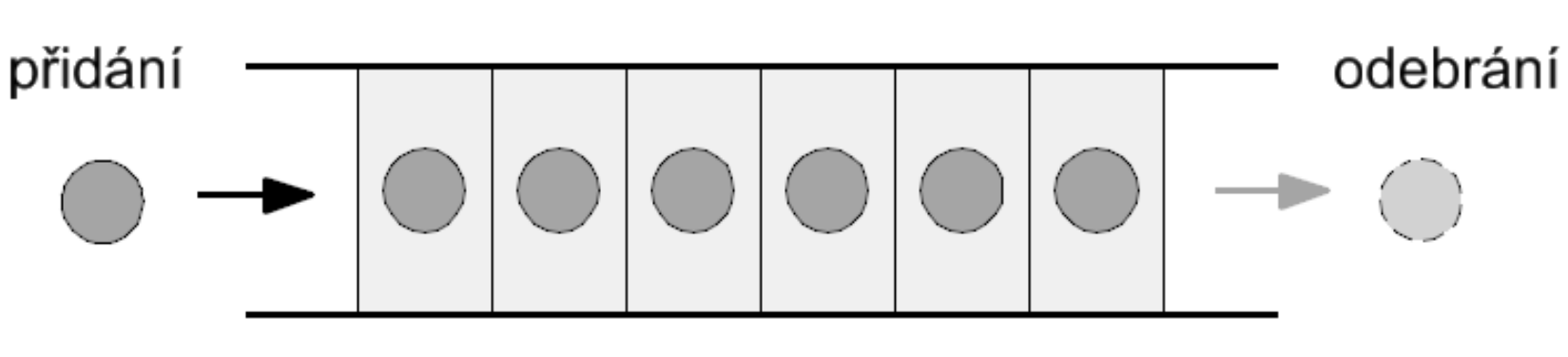
```
Q.put(1)
Q.put(2)
Q.put(3)
```

Odstranění prvku z fronty:

Použití metody `get()`, odebíráme z počátku fronty.

```
Q.get()
```

31. Ukázka fronty



32. Prioritní fronta (Priority Queue)

Nazývána fronta s předbíháním.

Modifikace fronty, u které hraje roli priorita prvku.

Uchována dvojice

$$\langle w, \text{element} \rangle ,$$

kde w, $w \in \mathbb{R}^+$, je priorita (váha) prvku.

Priorita přiřazena ohodnocovací funkcí.

Prvek s vyšší prioritou může přeběhnout prvek s nižší prioritou.

Princip prioritní fronty:

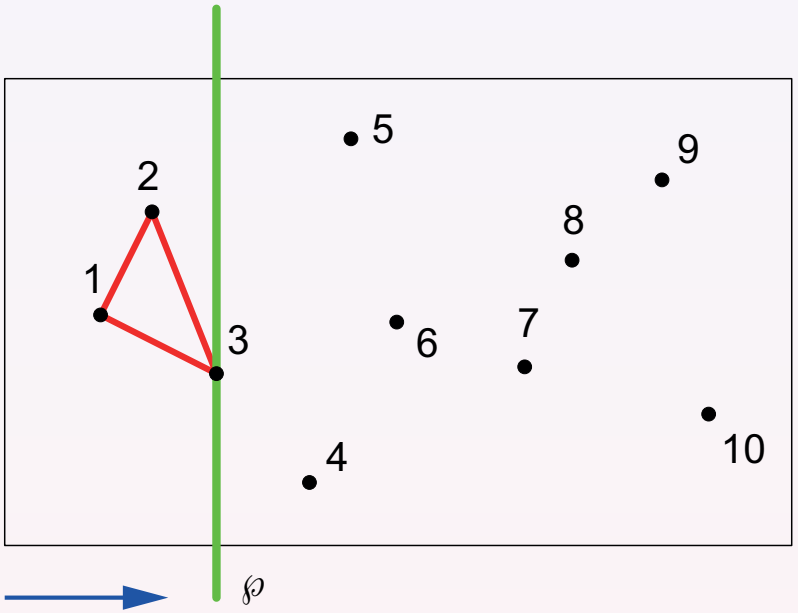
1. Prvky přidávány PQ v pořadí, v jakém jsou na vstupu.
2. Prvky odebírány z PQ na základě w (prvek s w_{max} první).
3. Pokud mají prvky stejnou prioritu, odebírány dle pořadí přidání do PQ.

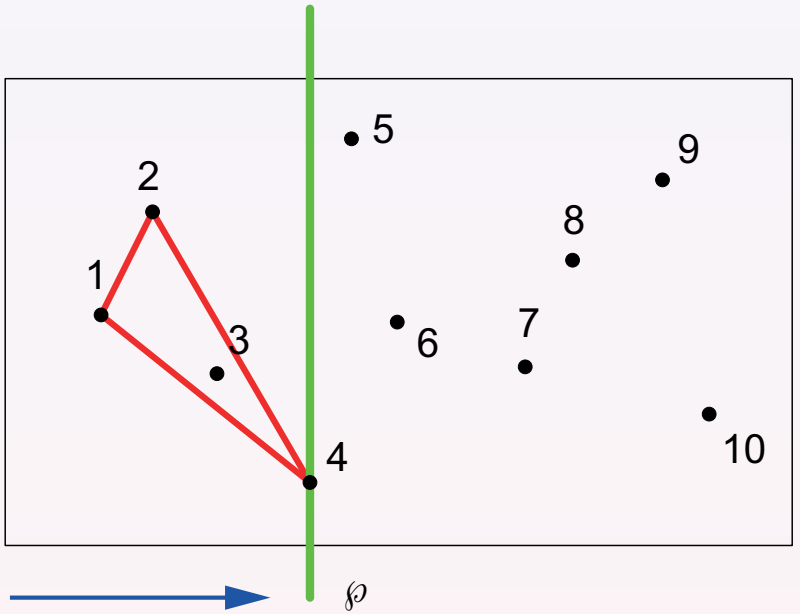
Časté použití v počítačové grafice, geoinformatice, ...

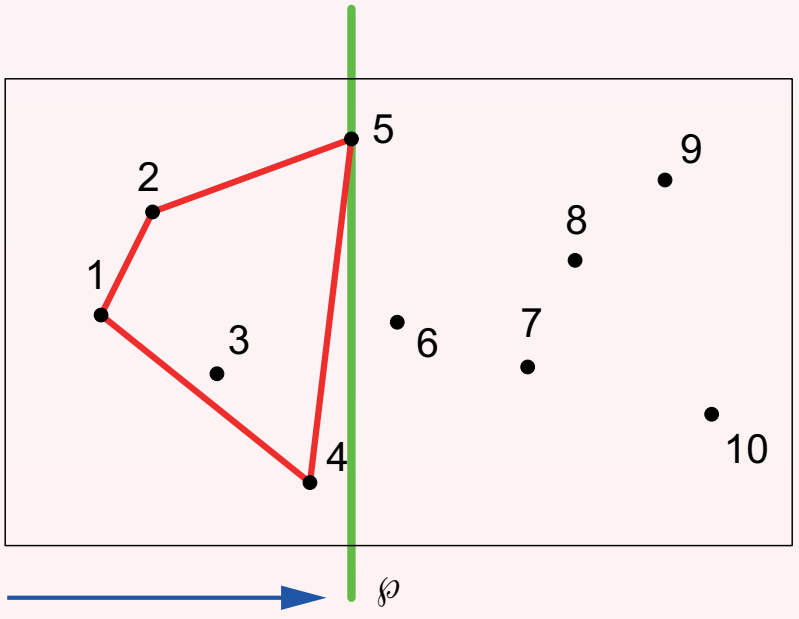
Realizace inkrementální strategie: Sweep Line (zametací přímka).

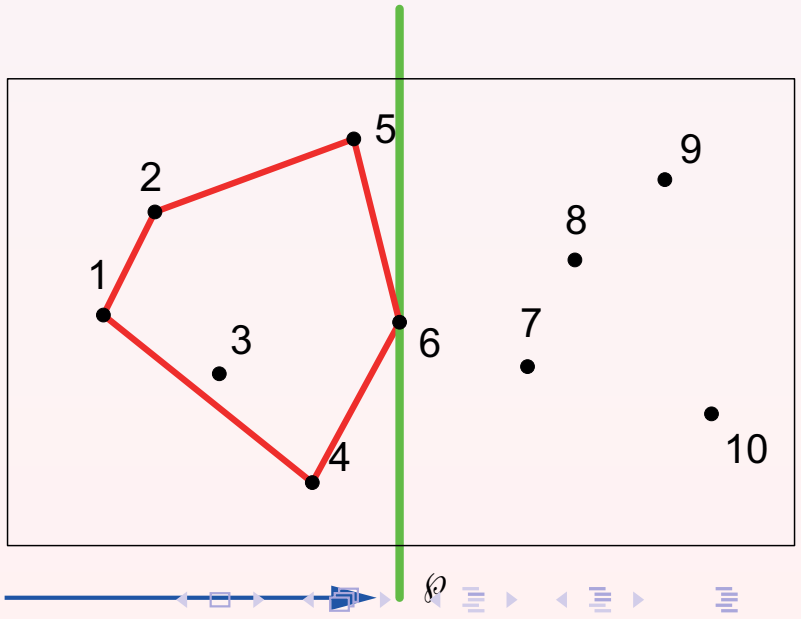
33. Sweep Line, konstrukce konvexní obálky

Body v prioritní frontě dle souřadnice x, seřazení zleva do prava.









34. Prioritní fronta v Pythonu

Python používá implementaci prioritní fronty na bázi haldy.

Vytvoření prioritní fronty:

```
import queue
Q = queue.PriorityQueue() #Volba typu fronty, prioritni
```

Přidání prvku do prioritní fronty:

Použití metody `put()`, přidání na konec fronty.

```
Q.put((2, “Tuesday”))
Q.put((1, “Monday”))
Q.put((3, “Wednesday”))
```

Odstranění prvku z priorotní fronty:

Použití metody `get()`, odebíráme z počátku fronty.

```
Q.get()
>>> (1, 'Monday')
```

Další metody:

Počet prvků ve frontě: metoda `qsize()`.

Test, zda je prázdná: metoda `empty()`.

35. N-tice (Tuple)

Obdoba seznamu včetně podporovaných operací.

Jednotlivé prvky n-tic na rozdíl od seznamů *nelze modifikovat* (immutable).

Prvky “konstantní”, chráněny proti zápisu.

Výhodou vyšší rychlost.

Často používány jako návratové parametry funkcí.

L[-7]	L[-6]	L[-5]	L[-4]	L[-3]	L[-2]	L[-1]		
123	456.3	”Hello”	-37.3	”World”	False	-1	”PC”	False
L[0]	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]		

Položky n-tice uzavřeny v kulatých závorkách.

```
L = (123, 456.3, -37.3) #N-tice se 3 polozkami
```

N-tice mohou být i vnořené.

```
L = (123, 456.3, ”Hello”, -37.3, ”World”, False, (-1, “PC”, False))
L[0] == L[-7] = 123
L[2][1] == L[-5][-4] = “e”
K = L[2:4]    #Rez
n = len L     #Pocet prvku
```

Nemají k dispozici žádné modifikační metody:

```
append(), insert(), pop(), del().
```

36. Množina (Set)

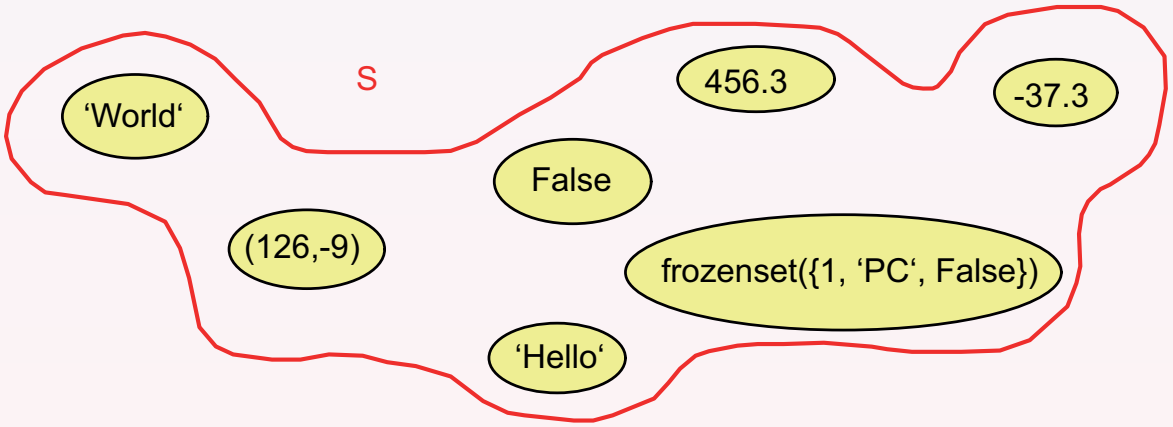
Neuspořádaná množina unikátních objektů (nemohou být duplicitní).

Unikátnost zajištěna hashovací funkcí h

$$(a_i, x_i), \qquad a_i = h(x_i).$$

K prvkům nelze přistupovat přímo (index), nelze provádět řezy.

Prvky množiny lze modifikovat.



Položky množiny uzavřeny v lomených závorkách.

Množiny mohou být vnořené: tvořeny n-ticemi, podmnožinami (frozensets).

```
S = {(-126,9), 456.3, ''Hello'', -37.3, ''World'', False,
     frozenset(-1, ''PC'', False)}  #Vnorena mnozina
L = (1,2,3)
S = set(L)  #Mnozina z entice
```

37. Vlastnosti množin

V Pythonu implementovány s využitím Hash Table.

Složitost operací $O(N)$.

Ve většině případů výkonnější než seznam.

\+ Přidávání prvku do množiny $\Theta(1)$.

\+ Hledání prvku v množině $\Theta(1)$.

\+ Mazání prvku $\Theta(1)$.

\- V nepříznivých případech kolize: operace nemají konstantní složitost.

\- Nastává pro “nevhodná” data.

Použití množin:

Použití pro práci s velkými daty.

V průměrném případě mnohem efektivnější než jiné datové struktury (hledání).

Umožňují booleovské operace s prvky.

38. Základní operace s množinami

Vytvoření prázdné množiny:

```
S = {}     S = set()
```

Délka seznamu: příkaz `len()`

```
n = len(S)
```

Přidání prvku: metoda `add()`

```
S.add(-126.9)
```

Přidání jiné podmnožiny: metoda `insert()`

```
S.insert({456.3,-137.3,"Hello"})
```

Nalezení položky na pozici v množině: funkce `in`

```
r = -137 in S #True nebo False
```

Odstranění posledního prvku z množiny: metoda `pop()`

```
n = S.pop()
```

Odstranění položky *p* z množiny: metody `remove(), discard()`

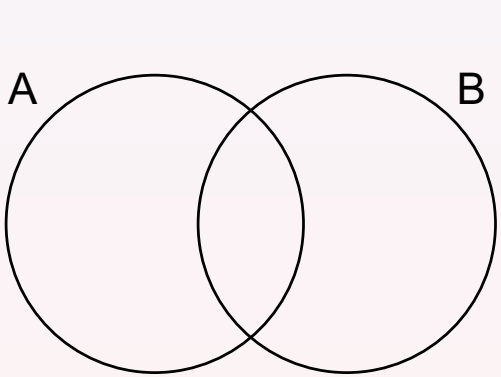
```
S.remove(-137)  #Pokud S neobsahuje p, vyjimka
S.discard(-137) #Vyjimka neprobehne
```

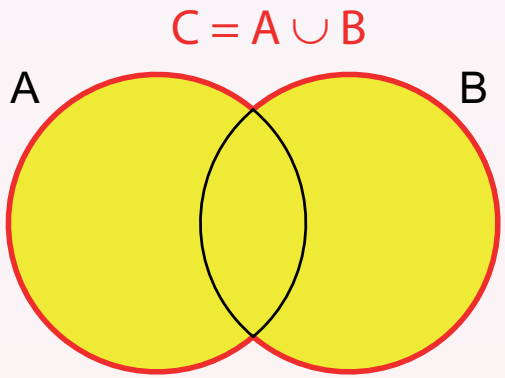
39. Množinové operace v Pythonu

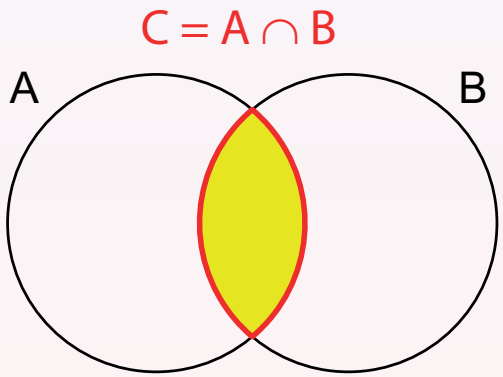
4 základní operace booleovské operace s množinami:

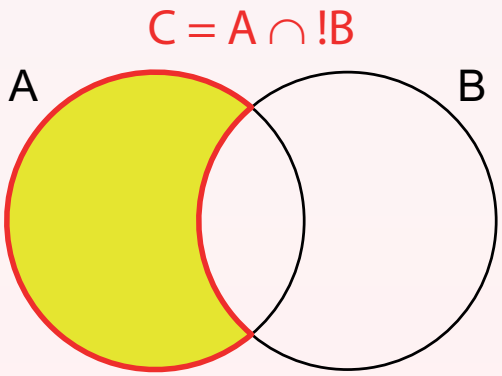
- Union: $C = A \cup B$.
- Intersection: $C = A \cap B$.
- Difference: $C = A \cap \overline{B}$, $B \cap \overline{A}$.
- Symmetric Difference: $C = (A \cup B) \cap \left(\overline{A \cap B}\right)$.

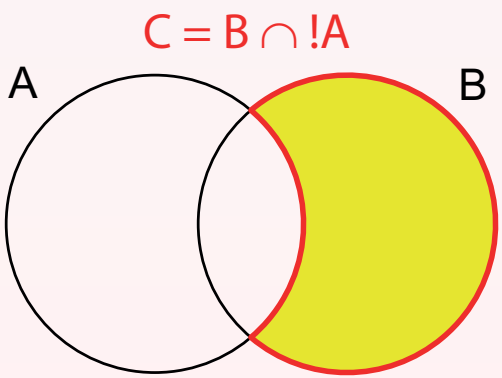
Konjunkce $\cap$, disjunkce $\cup$, negace $\bar{\cdot}$ resp !.

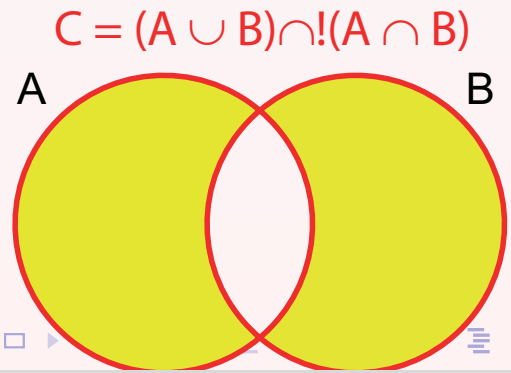












40. Sjednocení (Union)

Komutativita (symetrie):

$$A \cup B = B \cup A.$$

Asociativita

$$A \cup (B \cup C) = (A \cup B) \cup C.$$

Příklad:

```
A = {"H", "e", "l", "l", "o"}
B = {"w", "o", "r", "l", "d"}
C = A.union(B))
>>> {'d', 'o', 'H', 'e', 'w', 'r', 'l'}
```

41. Průnik (Intersection)

Komutativita (symetrie):

$$A \cap B = B \cap A.$$

Asociativita

$$A \cap (B \cap C) = (A \cap B) \cap C.$$

Příklad:

```
A = {"H", "e", "l", "l", "o"}
B = {"w", "o", "r", "l", "d"}
C = A.intersection(B))
>>> {'o', 'l'}
```

42. Rozdíl (Difference)

Neplatí komutativita

$$A \ominus B \neq B \ominus A, \qquad A \cap \overline{B} \neq B \cap \overline{A},$$

ani asociativita

$$A \ominus (B \ominus C) \neq (A \ominus B) \ominus C, \qquad A \cap (\overline{B \cap \overline{C}}) \neq (A \cap \overline{B}) \cap \overline{C}.$$

Avšak

$$A \ominus (B \cap C) = (A \ominus B) \cup (A \ominus C), \qquad A \cap (\overline{B \cap C}) = (A \cap \overline{B}) \cup (A \cap \overline{C}),$$
$$A \ominus (B \cup C) = (A \ominus B) \cap (A \ominus C), \qquad A \cap (\overline{B \cup C}) = (A \cap \overline{B}) \cap (A \cap \overline{C}),$$

Příklad:

```
A = {"H", "e", "l", "l", "o"}
B = {"w", "o", "r", "l", "d"}
C = A.difference(B))
D = B.difference(A))
>>> {'e', 'H'}
>>> {'w', 'd', 'r'}
```

43. Symmetric Difference (XOR)

Geometrická představa: sjednocení - průnik.

Komutativita (symetrie):

$$A \triangle B = B \triangle A \qquad (A \cup B) \cap \left(\overline{A \cap B}\right) = (B \cup A) \cap \left(\overline{B \cap A}\right).$$

Asociativita

$$A \triangle (B \triangle C) = (A \triangle B) \triangle C,$$

kde

$$A \triangle (B \triangle C) = (A \cup [(B \cup C) \cap \overline{(B \cap C)}]) \cap \overline{(A \cap [(B \cup C) \cap \overline{(B \cap C)}]},$$

$$(A \triangle B) \triangle C = ([(A \cup B) \cap (\overline{A \cap B})] \cup C) \cap \overline{([(A \cup B) \cap (\overline{A \cap B})] \cap C)}.$$

Příklad:

```
A = {"H", "e", "l", "l", "o"}
B = {"w", "o", "r", "l", "d"}
C = A.symmetric_difference(B))
>>> {'r', 'w', 'd', 'e', 'H'}
```

44*. Frozen Sets

Množina, kterou nelze měnit, tj. read only.

Tvořena immutable objekty.

Analogie s enticemi.

Lze ji vytvořit jak ze seznamu a množiny.

Používány často jako podmnožiny.

Vytvoření Frozen Setu ze seznamu:

```
S = frozen_set([values])
```

Vytvoření frozen setu z množiny:

```
S = frozen_set({values})
```

Podporují stejné operace jako “běžné” množiny.

45. Slovník (Dictionary)

Neuspořádaná množina unikátních dvojic

```
(klíč : hodnota)
```

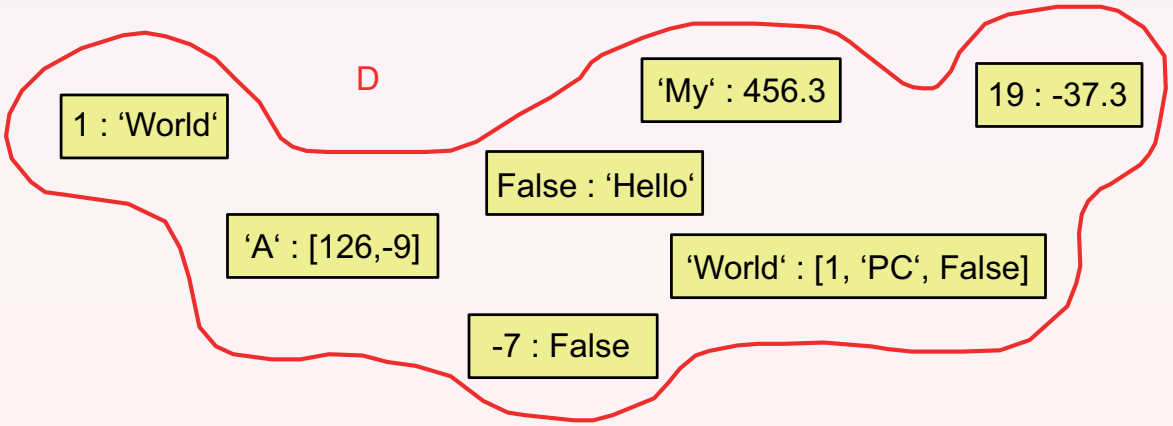
Jako klíč lze použít pouze *immutable objekty*.

Klíč unikátní v celém seznamu.

Hodnota může být reprezentována libovolným typem.

U dvojice (záznamu) lze měnit pouze hodnotu.

Implementován s využitím *hashovací tabulky,* operace v $\Theta(1)$.



Použití: rychlé vyhledání hodnoty dle klíče.

V jiných programovacích jazycích známa jako map (unordered set).

46. Operace se slovníkem

Vytvoření slovníku:

```
D = {1 : "World", -7 : False, 3 : "Wednesday", 19 : -37.3, 'A' : [-126, 9],
    'My' : 456.3, 'False' : Hello, 'World' : [1, 'PC', False]}
```

Délka slovníku: funkce `len()`

```
n = len(D)
```

Přidání prvku: metoda `add()`

```
D[13] = 'New value'
```

Nalezení položky ve slovníku: funkce `in`

```
r = -137 in S #True nebo False
```

Nalezení hodnoty dle klíče: metoda `get()`

```
r = D.get(13)
```

Odstranění položky ze slovníku: funkce `del()`

```
del(D, -137)
```

Procházení slovníku:

```
for d in D          #Prochazeni po klicich
for d in D.values() #Prochazeni po hodnotach
for d in D.items()  #Prochazeni po klicich i hodnotach
```

Slovník podporuje všechny *množinové operace*.

47. Dynamické datové struktury a typová kontrola

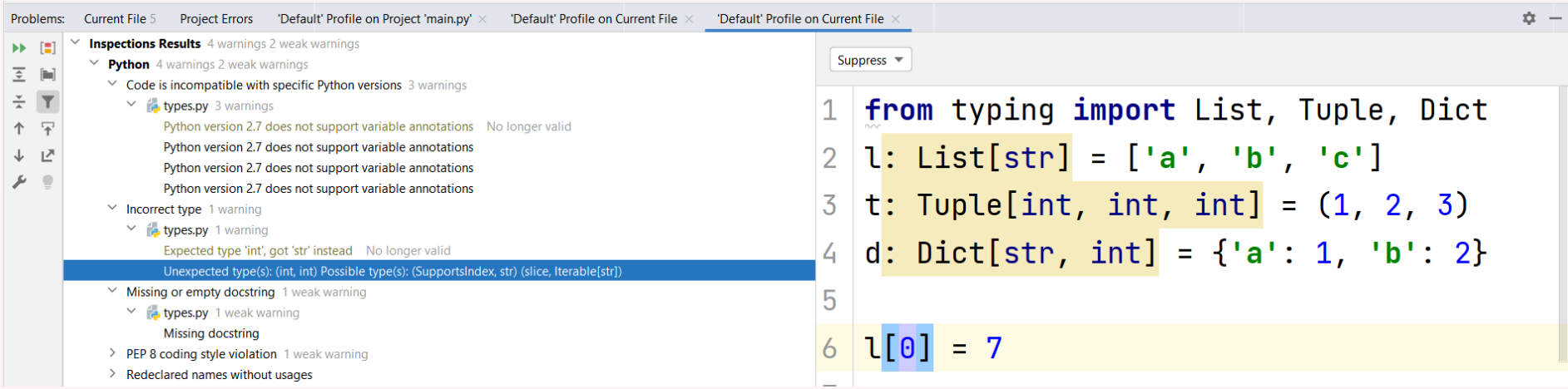
Podpora Type Hints i pro dynanamické datové struktury.

Tyto informace lze použít pro typovou kontrolu.

```
from typing import List, Tuple, Dict
l: List[str] = ['a', 'b', 'c']                  #Seznam objektu typu string
t: Tuple[int, int, int] = (1, 2, 3)             #N-tice objektu typu int
d: Dict[str, int] = {'a': 1, 'b': 2, 'c' :3}  #Slovnik, key=string, val =  int
```

Typová kontrola, odhalení chyby:

```
l[0] = 7                #Prirazeni objektu typu int, ocekavano str
```



48*. Hashing

Hash = otisk.

Využívá hashovací funkci h

$$a = h(k).$$

Transformuje klíč k libovolné délky na řetězec bitů a pevné délky.

Distribuce a po intervalu prováděna “inteligentně” (rovnoměrně).

Volena tak, aby:

- z a nebylo možné zpětně určit k,
- minimalizovala situace $k_1 \neq k_2$, kdy $h(k_1) = h(k_2)$.

Jinak vznik **kolize**: různé klíče transformovány na stejnou adresu.

Lze obejít dodatečnými úpravami.

Efektivita závisí na délce klíče (< klíč, > kolizí).

Nepodporuje některé důležité operace: sort.

Použití hashovacích funkcí:

- Datové struktury: hashovací tabulka (pouze unikátní klíče).
- Kryptografie: digitální podpis, kontrolní součty (MD5).
- Jedna z nejefektivnějších technik vyhledávání $\Theta(1)$.

49*. Hashovací funkce

Hashovací funkce (jednosměrná)

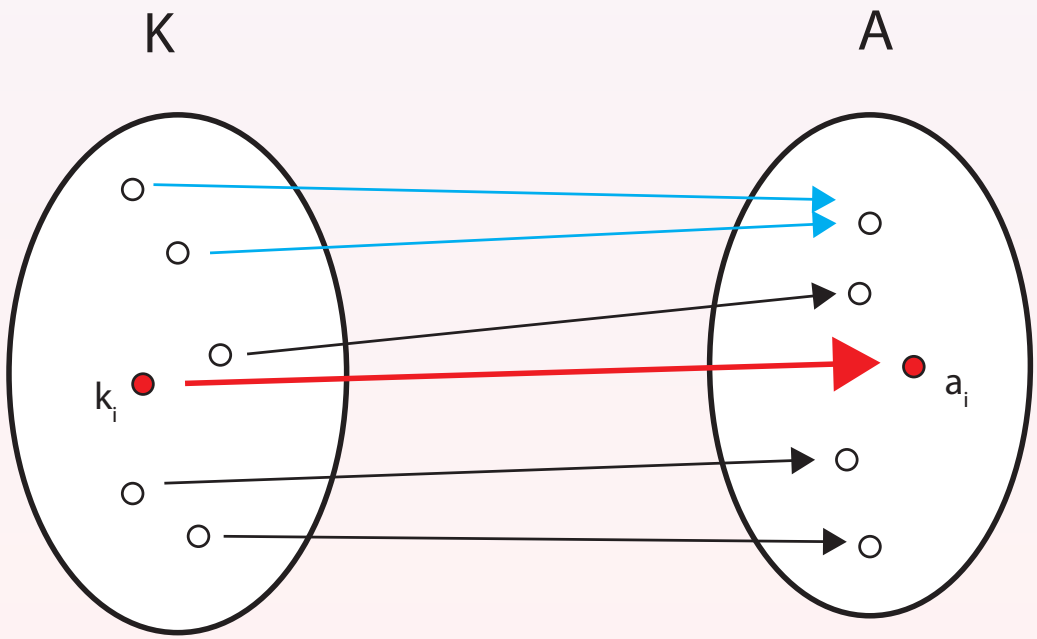
$$h : K \to A, \qquad h(k) = a, \qquad \|A\| \ll \|K\|, \qquad k \in K, a \in A.$$

Prostor klíčů K zobrazen do prostoru adres A. Klíč k posloupnost bitů libovolné délky.

Adresa a posloupnost bitů pevné délky.

Pro $\forall a \in A$ snadný výpočet $a = h(k)$.

Pro $a \in A$ velmi obtížné najít $k \in K$, neumíme rychle/vůbec $k = h^{-1}(a)$.



Protože $\|A\| \ll \|K\|$ ⇒ kolize.

50**. Matematický základ (1/2)

Dvě soudělná čísla $a, b \in \mathbb{N}$

$$a|b, \qquad a \text{ dělí } b, \qquad b = k \cdot a, \qquad k \in \mathbb{N}.$$

Nesoudělná čísla a, b, společný dělitel

$$(a, b) = 1.$$

Zbytek c po dělení a číslem b, operace modulo

$$a \equiv c \pmod{b}.$$

Příklad: $255 \equiv 7 \pmod{8}$, protože

$$\begin{array}{rcl} 255 & : & 8 = 31 \\ 15 & & \\ 7 & & \end{array}$$

Prvočíslo p dělí $a \cdot b$

$$a \cdot b \equiv 0 \pmod{p} \Leftrightarrow a \equiv 0 \pmod{p} \vee b \equiv \pmod{p}.$$

Příklad: $a = 3$, $b = 10$, $p = 5$, pak $3 \cdot 10 \equiv 0 \pmod{5}$, $10 \equiv 0 \pmod{5}$.

Malá Fermatova věta:

Jestliže $a \in \mathbb{N}$, p je prvočíslo, pak

$$p|a^p - a, \quad \text{resp.} \quad a^{p-1} \equiv 1 \pmod{p}.$$

Příklad: $a = 4$, $p = 3$, pak $3|60$, $16 \equiv 1 \pmod{3}$.

51**. Matematický základ (2/2)

Řád $e = \mathrm{ord}_d a$ čísla a modulo d je nejmenší kladný exponent e splňující

$$d|a^e - 1 \quad \text{resp.} \quad a^e \equiv 1 \pmod{d}, \qquad (a, d) = 1$$

Příklad: $\mathrm{ord}_5 3 = 5|3^e - 1$, pak $5|3^4 - 1$, $e = 4$, $\mathrm{ord}_7 2 = 7|2^e - 1$, pak $7|2^3 - 1$, $e = 3$.

Pokud d prvočíslem, pak

$$e = \mathrm{ord}_p a \leq p - 1.$$

Primitivní kořen modulo p

$$g^k \equiv 1 \pmod{p}, \qquad k = \{1, ..., p - 2\}, \qquad \mathrm{ord}_p g = p - 1.$$

Pro p existuje g takové, že pro všechny exponenty k je $g^k - 1$ nesoudělné s p.
Příklad: $p = 17$, $g = 3$, pak $3^k \equiv 1 \pmod{1}7$. Ale $3^{16} \equiv 1 \pmod{1}7$, $e = 16$.

Je-li p prvočíslo, pak **existuje** primitivní kořen modulo g.

Jednosměrná funkce $f : A \to B$:
Pro $\forall x \in A$ snadný výpočet $y = f(x)$.
Pro $y \in B$ velmi obtížné najít $x \in A$, neumíme rychle/vůbec $x = f^{-1}(y)$.
Využití v moderní kryptografii

$$f(x) \equiv g^x \pmod{p},$$

p velké prvočíslo a, x celočíselná hodnota.

52*. Kolize hashovací funkce

Může nastat, pokud

$$h(k_1) = h(k_2), \qquad k_1 \neq k_2.$$

Důsledek: více klíčů může mít stejné adresy.

Problém kolize není možné odstranit: $\|A\| \ll \|K\|$.

Vhodnou volbou h lze snížit pravděpodobnost výskytu kolize.

U některých problémů (kryptografie) vede k selhání metody.

Požadavky na hashovací funkci:

1) jednoduchost výpočtu: $h(k)$ polynomiální čas,
2) malá časová/paměťová režie,
3) aproximace náhodné funkce (rovnoměrné rozložení a_i pro k_i),
4) rezistence vůči kolizím,
5) rezistence vzoru: $k \neq h^{-1}(a)$,
6) deterministická.

53*. Ukázky hashovacích funkcí (1/2)

Modulo hashování

$$h(k) = k\%M.$$

Nejjednodušší varianta hashování.

Volba M: $M \neq 2^p$, lépe prvočíslo, např. $M = 31, 97$.

Čím menší M, tím více kolizí.

k	$(k)_2$	$k\%100$	$k\%16$	$k\%97$
12873	11001001001001	73	$2^3 + 2^0 = 9$	69
1073	10000110001	73	$2^0 = 1$	6
173	10101101	73	$2^3 + 2^2 + 2^0 = 13$	76
135	10000111	35	$2^2 + 2^1 + 2^0 = 7$	38
263	100000111	63	$2^2 + 2^1 + 2^0 = 7$	69
247	11110111	47	$2^2 + 2^1 + 2^0 = 7$	53

Při převodu $(k_{10}) \to (k_2)$, kdy $M = 2^p$, je modulem posledních p bitů!

54*. Ukázky hashovacích funkcí (2/2)

Multiplication hashing, ukázky

$$\begin{aligned} h(k) &= (c * k)\%M, \qquad c = 16161, \\ h(k) &= (int)(c * (float)k)\%M, \qquad c = 0.616161, \\ h(k) &= \frac{M}{C}((d \cdot k)\%M), \qquad C = 2^{32}, d = 2654435769. \end{aligned}$$

Mildsquare hashing

$$h(k) = \frac{M}{C}(k^2\%M), \qquad \frac{C}{M} = 2^{C-p}, M = 2^p.$$

String hashing

$$h(k) = (c \cdot h(k) + h[i])\%M, \qquad h(k) = 0, c = 128.$$

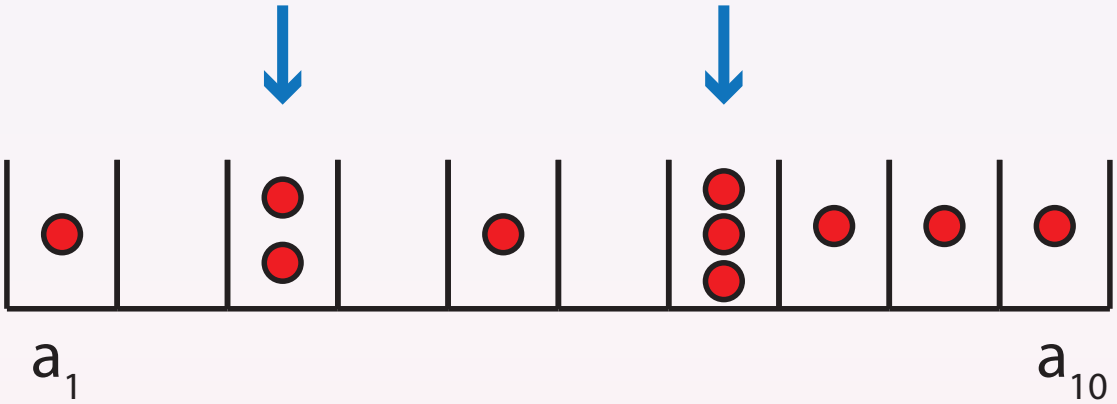
Neexistuje univerzální hashovací funkce.

Pro liché k liché $h(k)$ a naopak -> stejná pravděpodobnost.

55*. Odstranění kolizí

Kolizím se při hashování nelze vyhnout.

Jejich vliv na data lze částečně potlačit.



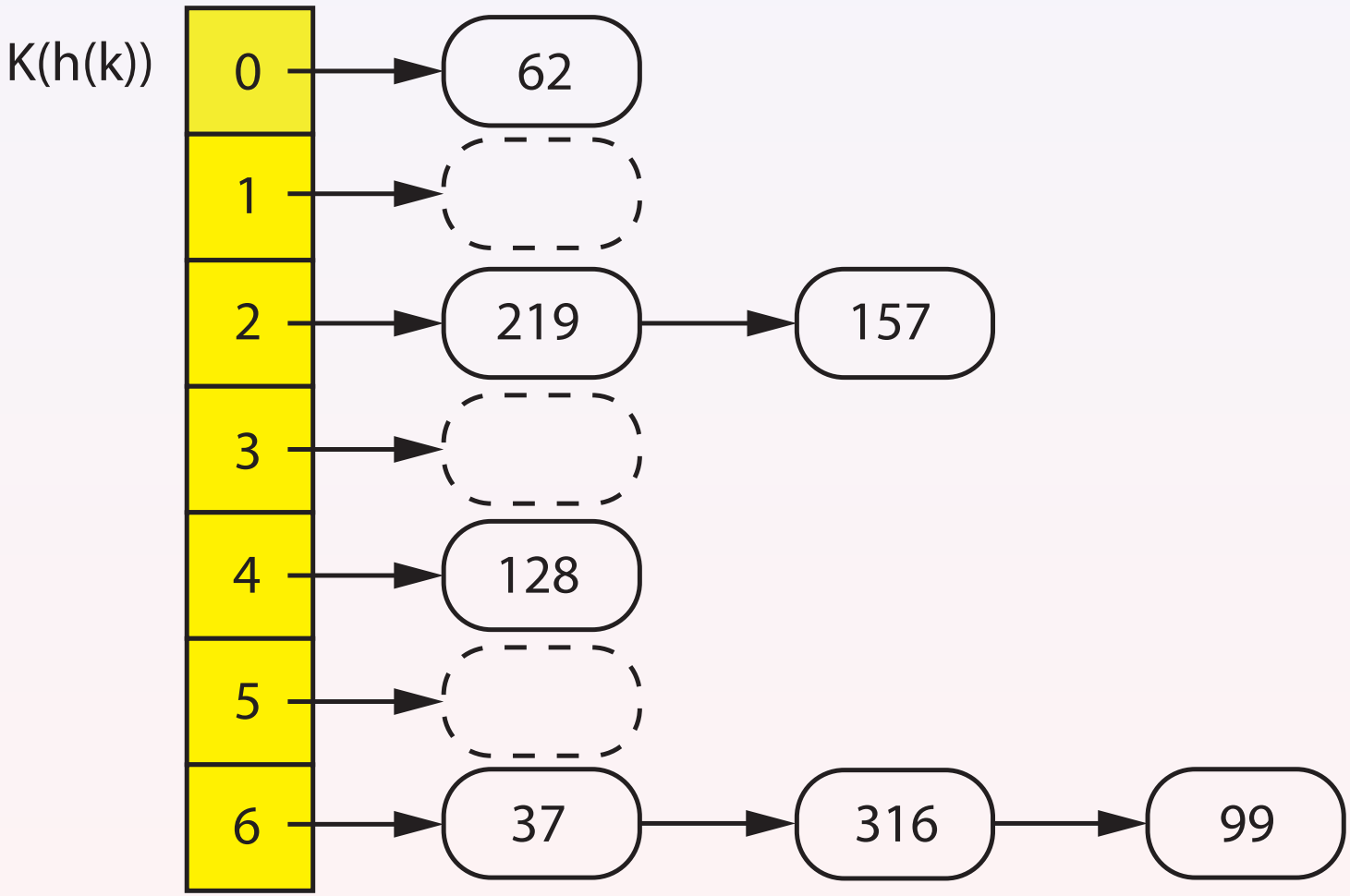
2 základní přístupy:

- akceptace kolize,
- minimalizace počtu kolizí.

Metody:

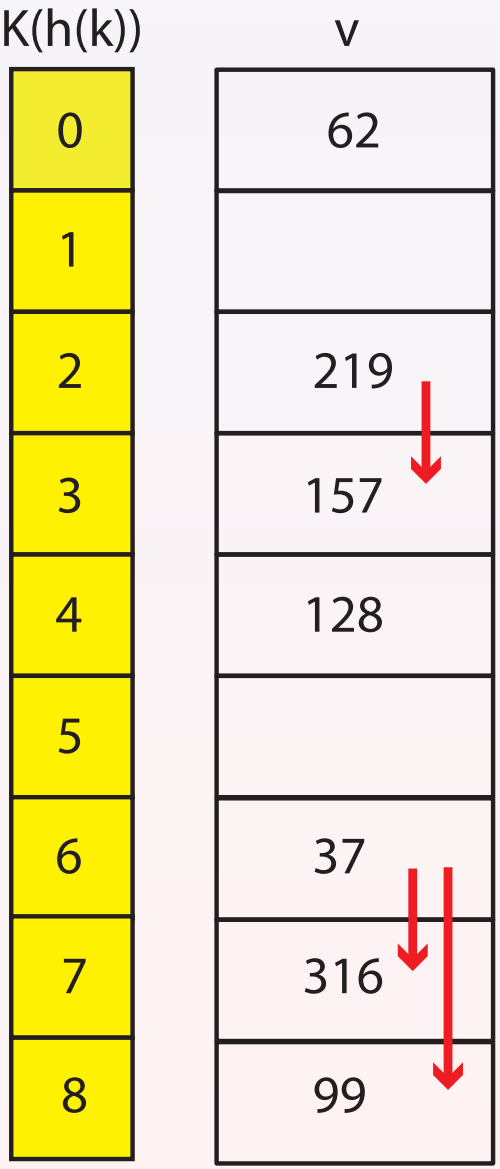
- Separate Chaining: kolidující buňka obsahuje lineární seznam.
- Linear Probing: duplicitní klíče ukládány do volných buněk.
- Double Hashing: LP + druhý hash $h_2(k)$ spočítá přírůstek v adrese.

56*. Ukázka Separate Chainingu



$$h(k) = k\%M, \qquad M = 31.$$

57*. Ukázka Linear Probingu

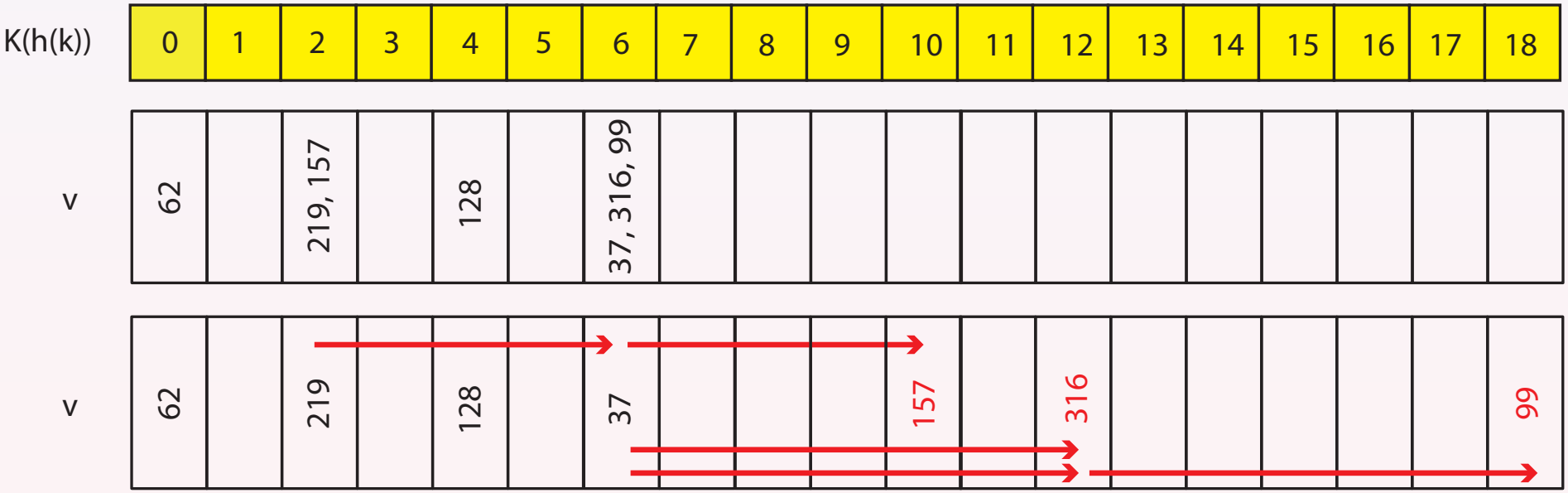


58*. Ilustrace Double Hashingu

Hashovací funkce:

$$h(k) = (h_1(k) + i \cdot h_2(k)) \,\%\, TS, \quad h_1(k) = k\%TS, \quad h_2(k) = PN - k\%PN,$$

kde TS je velikost tabulky, PN malé prvočíslo: $PN < TS$.



Příklad: $TS = 31$, $PN = 7$.

$$h(k) = (h_1(k) + i \cdot h_2(k)) \,\%\, TS, \quad h_1(k) = k\%31, \quad h_2(k) = 7 - k\%7.$$

$h_2(157) = 4$, $h_2(316) = 6$, $h_2(99) = 6$.