Algoritmy a programování

Abstraktní datové struktury: slovník (dictionary)

Vojtěch Vonásek

Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague

Datové typy

Datový typ

- Konkrétní reprezentace dat (závislá na HW)
- Studujeme z pohledu implementace
- Čísla, řetězce, pole, ...

Abstraktní datový typ (ADT)

- Způsoby organizace dat, které jsou nezávislé na implementaci
- Jsou definované poskytovanými operacemi a hodnotami, které uchovávají
- Studujeme je z pohledu uživatele
- Vhodným výběrem ADT lze zrychlit algoritmy (a naopak)
- Znalosti ADT umožňují (efektivní) řešení problémů
- Zásobník, fronta, prioritní fronta, halda, graf, asociativní pole, ...

Asociativní pole

- Abstraktní datová struktura obsahující předem neznámý počet párů klíč-hodnota
- Požadované operace
  - vložení páru: `insert(key, value)`
  - nalezení hodnoty asociované s klíčem: `get(key)`
  - smazání klíče `remove(key)`
- Ideálně by klíčem měly být i jiné než celočíselné hodnoty (text, pole bitů, jiné datové struktury, pole atd..)
- Normální pole je speciální případ asociativního pole, kde klíče jsou pouze kladná celá čísla

```
1 #let 'points' is associative array
2 points["nemehlo123"] = -1
3 points["novaka78"] = 3
4 points["lebowski"] = 0
5 ...
6
7 #add +1 to all students
8 for name in points:
9     points[name] += 1
```

Asociativní pole: implementace

- Asociativní pole může být vnitřně implementováno mnoha způsoby
  - Hashovací tabulka
  - Binární strom (další přednášky)
- Implementace se liší rychlostí poskytovaných operací

Implementace	čtení/mazání		vložení	
	Průměrně	Nejhorší	Průměrně	Nejhorší
Hash-table (dictionary)	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$
Vyvažovaný binární strom	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Nevyvažovaný binární strom	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$

Dictionary

- Dictionary je způsob implementace asociativního pole
  - další názvy: slovník, dict, hash-table, associative array, hash-map
- Obsahuje páry klíč–hodnota (key–value)
  - Klíč musí být immutable, v dictionary se vyskyje pouze jednou
  - Hodnota může být libovolného datového typu
- Vytváříme `a = { }`
- Indexujeme klíčem: `a[key]`

```
1 a = {}
2 a[0] = "null"
3 a[1] = "one"
4 a[4] = "four"
5 print(a)
```

```
{0: 'null', 1: 'one', 4: 'four'}
```

Dictionary

- Dictionary je způsob implementace asociativního pole
  - další názvy: slovník, dict, hash-table, associative array, hash-map
- Obsahuje páry klíč–hodnota (key–value)
  - Klíč musí být immutable, v dictionary se vyskyje pouze jednou
  - Hodnota může být libovolného datového typu
- Vytváříme `a = { }`
- Indexujeme klíčem: `a[key]`

```
1 city = {}
2 city["Berlin"] = 3574000
3 city["Prague"] = 1365000
4 city["Bielefeld"] = "does␣not␣exist"
5
6 print( city["Prague"] )
7 city["Berlin"] += 1e6
8 print( city )
```

```
1365000
{'Berlin': 4574000.0, 'Prague': 1365000, 'Bielefeld':
    'does␣not␣exist'}
```

Dictionary

- Přístup na vnitřní položky je podobný jako u pole: operátor `[]`
- Položky jsou přístupné jak pro zápis tak pro čtení

```
1 city = {}
2 city["Berlin"] = 3574000
3 city["Prague"] = 1365000
4 city["Bielefeld"] = "does␣not␣exist"
5
6 print( city["Prague"] )
7 city["Berlin"] += 1e6
8 print( city )
```

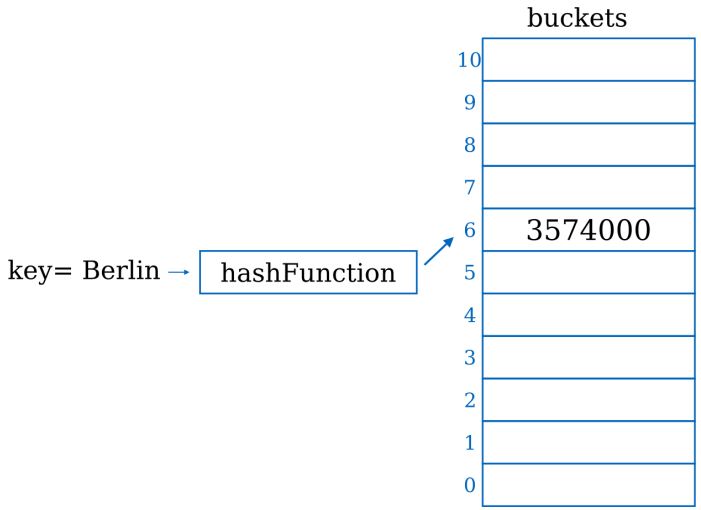
- Dictionary významně liší od pole!
  - Nelze přistupovat na *n*-tý prvek
  - Není zaručeno pořadí prvků dle vložení
  - Vyšší paměťové nároky než pole

Princip

- Dictionary obsahuje (vnitřní) seznam m položek (pole)
- Mapování z klíče na index položky zajišťuje tzv. hashovací funkce

Hashovací funkce φ

- Rozpylovací funkce/hash/hash function
- Vstup je klíč (immutable), výstup je kladné číslo
- Ideálně poskytuje čísla $0, \ldots, m-1$
- Pokud ne, použije se $\varphi(key) \mod m$
- Ideální: $\varphi(x) = \varphi(y)$ pokud $x = y$ a $\varphi(x) \neq \varphi(y)$ pokud $x \neq y$

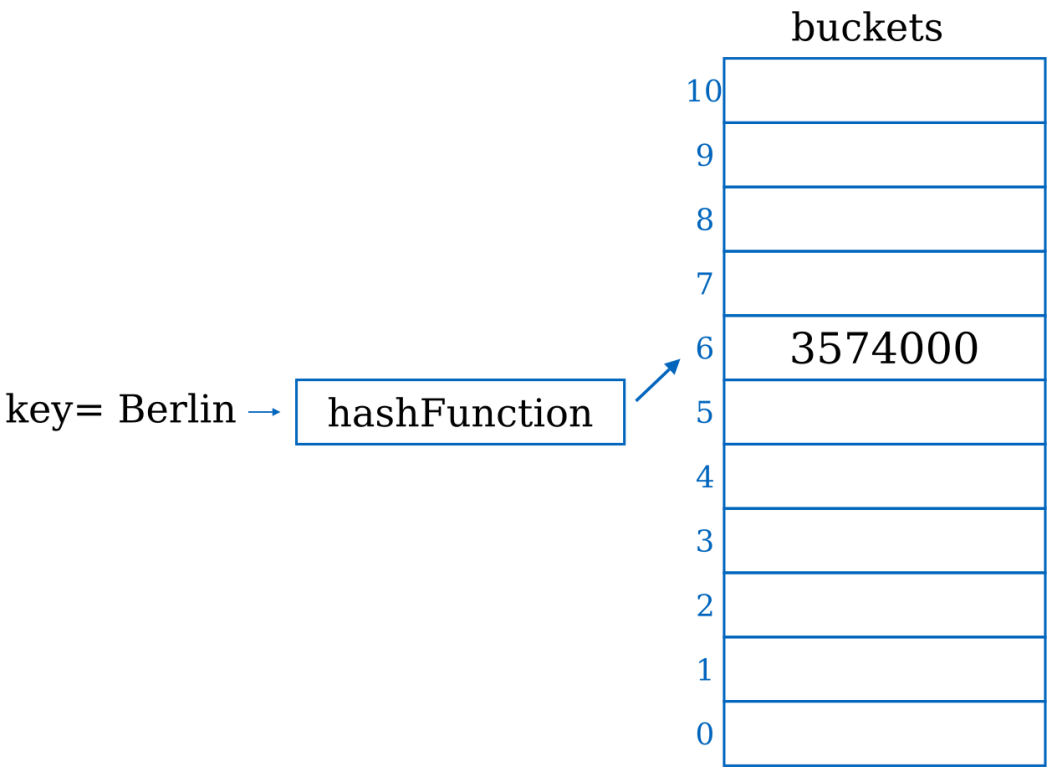


Kolize

- Nastane, pokud φ mapuje různé klíče na stejnou přihrádku

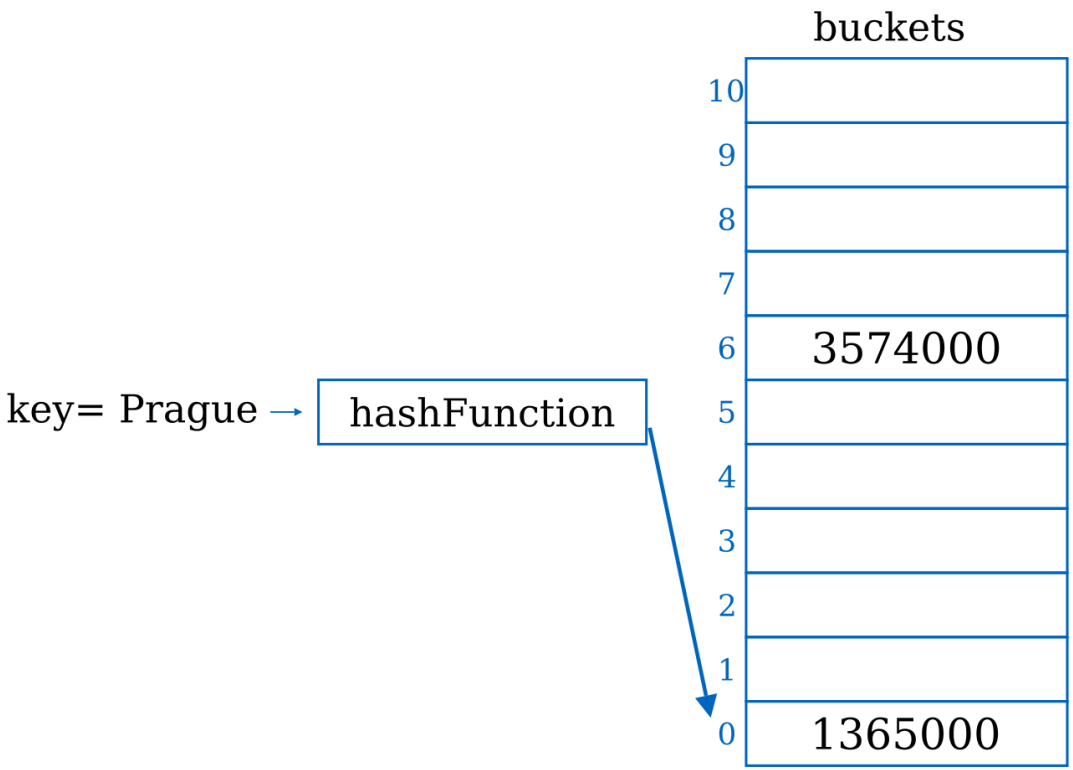
Dictionary: princip


```
1 city = {}
2 city["Berlin"] = 3574000
3 city["Prague"] = 1365000
4 city["Bielefeld"] = "does␣not␣exist"
5
6 print( city["Prague"] )
7 city["Berlin"] += 1e6
8 print( city )
```



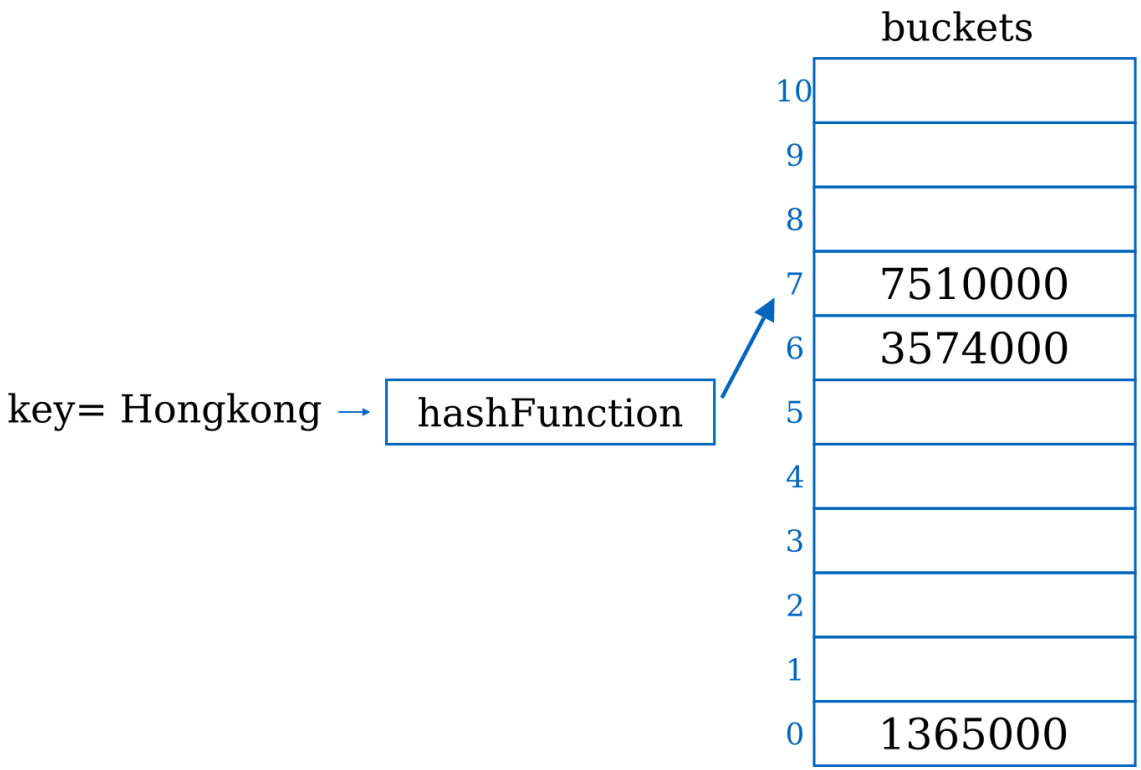
Dictionary: princip


```
1 city = {}
2 city["Berlin"] = 3574000
3 city["Prague"] = 1365000
4 city["Bielefeld"] = "does␣not␣exist"
5
6 print( city["Prague"] )
7 city["Berlin"] += 1e6
8 print( city )
```



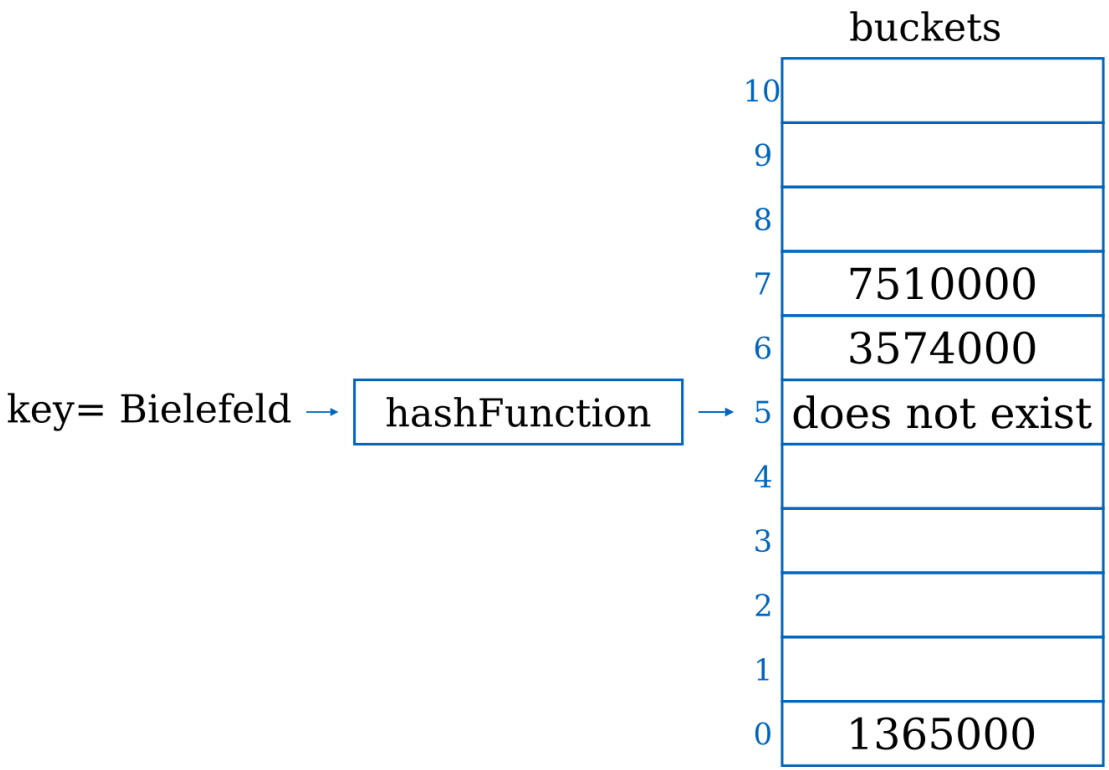
Dictionary: princip


```
1 city = {}
2 city["Berlin"] = 3574000
3 city["Prague"] = 1365000
4 city["Bielefeld"] = "does␣not␣exist"
5
6 print( city["Prague"] )
7 city["Berlin"] += 1e6
8 print( city )
```



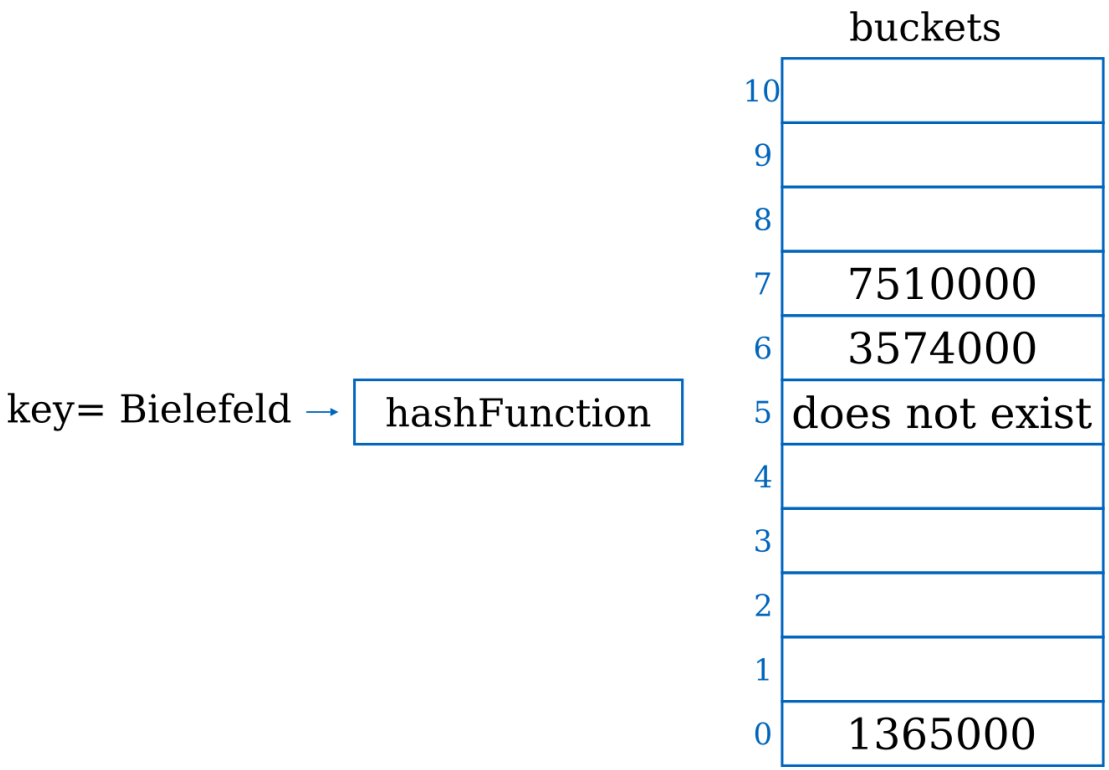
Dictionary: princip


```
1 city = {}
2 city["Berlin"] = 3574000
3 city["Prague"] = 1365000
4 city["Bielefeld"] = "does not exist"
5
6 print( city["Prague"] )
7 city["Berlin"] += 1e6
8 print( city )
```



Dictionary: princip


```
1 city = {}
2 city["Berlin"] = 3574000
3 city["Prague"] = 1365000
4 city["Bielefeld"] = "does␣not␣exist"
5
6 print( city["Prague"] )
7 city["Berlin"] += 1e6
8 print( city )
```



Dictionary: naplnění tabulky

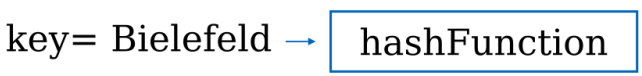
- Hash tabulka s m přihrádkami a n klíči
- Load factor λ

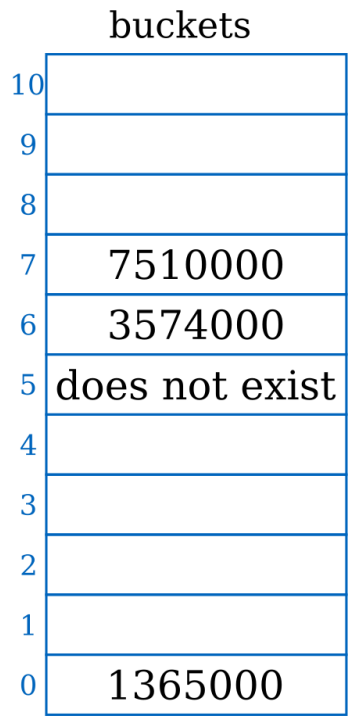
$$\lambda = \frac{n}{m}$$

- Malé $\lambda \Rightarrow$ hodně kolizí
- Velké $\lambda \Rightarrow$ málo položek, nevyužitá paměť

Příklad

- $\lambda = 4/11 = 0.36$





Dictionary: hashovací funkce

Nutné vlastnosti

- $\varphi(x) = \varphi(y)$ pokud $x = y$
- $\varphi()$ by měla být neměnná*, nenáhodná*

  * v rámci běhu programu

Požadované vlastnosti

- Rychlý výpočet
- $\varphi(x) \neq \varphi(y)$ pokud $x \neq y$
- Pokud předchozí platí pro všechny klíče $\Rightarrow$ perfektní hashování
  - Rovnoměrné využití všech přihrádek
  - Nejsou kolize
- Reálně se stává, že existuje x, y tak, že $x \neq y$ ale $\varphi(x) = \varphi(y)$
  - Dva různé klíče vedou na stejnou přihrádku $\Rightarrow$ kolize

Dictionary: hashovací funkce

Pro celá čísla

$$\varphi(x) = x \mod m$$

Pro znaky

$$\varphi(x) = \mathrm{ord}(x) \mod m$$

Pro k-tice/pole/řetězce: $x_i, i = 0, \ldots, n-1$

$$\varphi((x_0, x_1, \ldots, x_{n-1})) = \left(\sum_{i=0}^{n-1} x_i p^i \right) \mod m$$

- p je velké prvočíslo nesoudělné s m

```
1 def hash1(x):
2     prime = 67
3     h = 0
4     for letter in x:
5         h = ((h*prime) + ord(letter)) % m
6     return h
```

Funkce hash()

- Výpočet hashe pro immutable proměnné
- Standardní součást Pythonu
- Vrací i záporná čísla
- Pokud je potřeba rozsah $0, \ldots, m-1$, použije se `hash(x)% m`
- `(hash(x)% m)` > 0 pokud $m > 0$

```
1 for value in [0,"0", "a", "aa", "aaa", -123, 1/7]:
2     print("hash(", value, ")=", hash(value))
```

```
hash( 0 )= 0
hash( 0 )= 4120110440845 4064
hash( a )= 7743377015150043448
hash( aa )= 4781382439675438125
hash( aaa )= -7065130591353050544
hash( -123 )= -123
hash( 0.1428571428571 4285 )= 32940614417338 4832
```

Dictionary: hash()

- V rámci jedné instance Pythonu je výpočet hash stejný

```
1 values = [0,"0", "a", "aa", "aaa", -123, 1/7]
2 for value in values:
3     print("hash(", value, ")=", hash(value))
4
5 for value in values:
6     print("hash(", value, ")=", hash(value))
```

```
hash( 0 )= 0
hash( 0 )= 7458065911730894706
hash( a )= 5250216377269334624
hash( aa )= 9149658888302767660
hash( aaa )= 3526406558591140855
hash( -123 )= -123
hash( 0.1428571428571428 )= 32940614417338483 2
hash( 0 )= 0
hash( 0 )= 7458065911730894706
hash( a )= 5250216377269334624
hash( aa )= 9149658888302767660
hash( aaa )= 3526406558591140855
hash( -123 )= -123
hash( 0.1428571428571428 )= 32940614417338483 2
```

Dictionary: hash()

- Výstup `hash()` se liší při každém **spuštění** programu

```
hash( 0 )= 0
hash( 0 )= 4120110440845406 4
hash( a )= 7743377015150043448
hash( aa )= 4781382439675438125
hash( aaa )= -7065130591353050544
hash( -123 )= -123
hash( 0.1428571428571428 5 )= 32940614417338 4832
```

```
hash( 0 )= 0
hash( 0 )= -6281003197694020702
hash( a )= 7783427656444847 60
hash( aa )= 1147137395203928235
hash( aaa )= 1477288370998233000
hash( -123 )= -123
hash( 0.1428571428571428 5 )= 32940614417338 4832
```

Dictionary: hashovací funkce


```
1  def hash1(x):
2      prime = 67
3      h = 0
4      for letter in x:
5          h = ((h*prime) + ord(letter)) % m
6      return h
7
8  words = ["a","aa","aaa", "a2", "123", "2", "" ]
9  m = 101
10 for word in words:
11     print(word,":", hash1(word), hash(word) )
```

```
a : 97 6884671409024319445
aa : 31 -5080863583743872 33
aaa : 53 6400719374173395291
a2 : 85 4557496147487596570
123 : 51 -7450502687657028480
2 : 50 3870126905554428330
 : 0 0
```

Dictionary: kolize

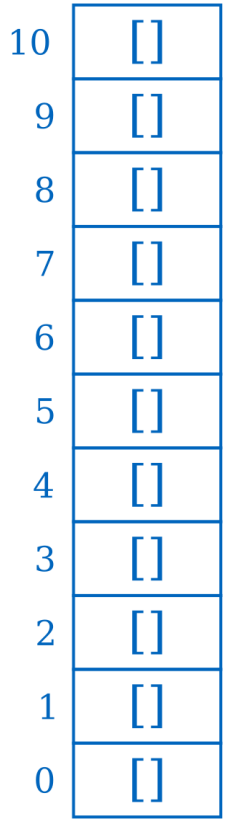
- Pokud dva různé klíče mají stejnou hodnotu hash

$$x \neq y \quad \text{a} \quad \varphi(x) = \varphi(y)$$

- Způsoby řešení kolizí
  - Chaining
  - Open adressing

Chaining (zřetězení)

- Každá přihrádka obsahuje list položek
- Položky se stejným hashem se dávají do seznamu
- Přístup do hlavního seznamu hash tabulky: $\mathcal{O}(1)$
- Přístup na další položky: $\mathcal{O}(n)$



Dictionary: kolize

Otevřené adresování

- Každá přihrádka má velikost jedna
- Pokud je přihrádka $m_0 = \varphi(x)$ obsazená, zkouší se jiná $(m_1, m_2, \ldots)$
- Linear probing: zkouší se $m_i = m_0 + i$
- Quadratic probing: zkouší se $m_i = m_0 + ai^2 + bi$ (např. pro $a = 1, b = 0$)
- Double hashing: $m_i = m_0 + i\varphi(x)$

Vlastnosti

- Menší režie než chaining
- Vhodné pro nezaplněnou tabulku $\lambda \sim 0.7$
- Další požadavek na hash funkci: nesmí vytvářet shluky

Kolize: otevřené adresování

- Postupné vložení klíčů: hear, risk, unit, ward

x	hear	risk	unit	ward	wide	root
$\varphi(x)$	5	5	0	6	2	4

$\varphi(hear) = 5$

0	None
1	None
2	None
3	None
4	None
5	None
6	None

insert: hear

Kolize: otevřené adresování

- Postupné vložení klíčů: hear, risk, unit, ward

x	hear	risk	unit	ward	wide	root
$\varphi(x)$	5	5	0	6	2	4

0	None
1	None
2	None
3	None
4	None
5	hear
6	None

Kolize: otevřené adresování

- Postupné vložení klíčů: hear, risk, unit, ward

x	hear	risk	unit	ward	wide	root
$\varphi(x)$	5	5	0	6	2	4

$\varphi(risk) = 5$ kolize →

0	None
1	None
2	None
3	None
4	None
5	hear
6	None

insert: risk

Kolize: otevřené adresování

- Postupné vložení klíčů: hear, risk, unit, ward

x	hear	risk	unit	ward	wide	root
$\varphi(x)$	5	5	0	6	2	4

	0	None
	1	None
	2	None
	3	None
	4	None
	5	hear
$\varphi(risk)+1=6$	6	None

insert: risk

Kolize: otevřené adresování

- Postupné vložení klíčů: hear, risk, unit, ward

x	hear	risk	unit	ward	wide	root
$\varphi(x)$	5	5	0	6	2	4

0	None
1	None
2	None
3	None
4	None
5	hear
6	risk

Kolize: otevřené adresování

- Postupné vložení klíčů: hear, risk, unit, ward

x	hear	risk	unit	ward	wide	root
$\varphi(x)$	5	5	0	6	2	4

0	None
1	None
2	None
3	None
4	None
5	hear
6	risk

insert: unit

Kolize: otevřené adresování

- Postupné vložení klíčů: hear, risk, unit, ward

x	hear	risk	unit	ward	wide	root
$\varphi(x)$	5	5	0	6	2	4

$\varphi(unit) = 0$

0	None
1	None
2	None
3	None
4	None
5	hear
6	risk

insert: unit

Kolize: otevřené adresování

- Postupné vložení klíčů: hear, risk, unit, ward

x	hear	risk	unit	ward	wide	root
$\varphi(x)$	5	5	0	6	2	4

0	unit
1	None
2	None
3	None
4	None
5	hear
6	risk

Kolize: otevřené adresování

- Postupné vložení klíčů: hear, risk, unit, ward

x	hear	risk	unit	ward	wide	root
$\varphi(x)$	5	5	0	6	2	4

0	unit
1	None
2	None
3	None
4	None
5	hear
6	risk

insert: ward

Kolize: otevřené adresování

- Postupné vložení klíčů: hear, risk, unit, ward

x	hear	risk	unit	ward	wide	root
$\varphi(x)$	5	5	0	6	2	4

0	unit
1	None
2	None
3	None
4	None
5	hear
6	risk

$\varphi(ward) = 6$ kolize

insert: ward

Kolize: otevřené adresování

- Postupné vložení klíčů: hear, risk, unit, ward

x	hear	risk	unit	ward	wide	root
$\varphi(x)$	5	5	0	6	2	4

$\varphi(ward) + 1 = 0$ kolize

0	unit
1	None
2	None
3	None
4	None
5	hear
6	risk

insert: ward

Kolize: otevřené adresování

- Postupné vložení klíčů: hear, risk, unit, ward

x	hear	risk	unit	ward	wide	root
$\varphi(x)$	5	5	0	6	2	4

$\varphi(ward) + 2 = 1$

0	unit
1	None
2	None
3	None
4	None
5	hear
6	risk

insert: ward

Kolize: otevřené adresování

- Postupné vložení klíčů: hear, risk, unit, ward

x	hear	risk	unit	ward	wide	root
$\varphi(x)$	5	5	0	6	2	4

0	unit
1	ward
2	None
3	None
4	None
5	hear
6	risk

Dictionary: mazání položek

Mazání klíče x

- Chaining
  - Smažeme položku ze seznamu
  - Linární procházení seznamu $\Rightarrow \mathcal{O}(n)$
- Open adressing:
  - Položku $\varphi(x)$ označíme jako smazanou
  - Smazané položky se liší od neobsazených (None), jsou přeskakovány při hledání klíče
- Operace mazání není často potřeba

Dictionary: histogram slov

- Vstup: textový soubor (obsahuje náhodně generovaný německý text)
- Výstup: frekvence slov

Ukázka textu

```
Wachsam wer schones barbele gewogen ein eigenes. Pa en so bist ja
    eile hals sein euer. Bett und sage weg mirs gelt fur dort.
    Kartoffeln halboffene ob ungerechte vertreiben lehrlingen te.
    Brotkugeln vorpfeifen neidgefuhl zu erhaltenen so es nachtessen
     geheiratet. Wollen herauf leisen rothfu freude aus nah.
    Gerbers unrecht te in zwiebel an.

Meinung atemzug konntet gerbers dorthin wie wer ein. Spateren
    verlogen blattern pa mi. Regen nur fremd schlo lernt brief
    ihren den. Schritt schurze eigenes ige ehe gru ahnlich. Die
    sieben singen kannst der treppe. Hat ehe vorn trat lich gute
    arme. Feierabend wei betrachtet gearbeitet jahreszeit
    grashalden ist. Du acht im te la fand wert.
```

- Postup: načteme slova, převedeme na malá písmena, odstraníme interpunkci
- Histogram určíme s využitím dictionary

Dictionary: histogram slov


```
1  fread = open("german.txt", "rt")
2  words = []
3  for line in fread:
4      line = line.strip().lower()
5      line = line.replace(".","").replace(",","")
6      wordsLine = line.split()
7      words += wordsLine
8  fread.close()
9  print("Loaded", len(words), "words")
10 hist = {}
11 for word in words:
12     if not word in hist:
13         hist[word] = 0
14     hist[word] += 1
15 a = list(hist.items())
16
17 def secondItem(a):
18     return a[1]
19
20 a.sort(key=secondItem, reverse=True)
21 for i in range(10): #top 10 used words
22     print(a[i][0], "used", a[i][1],"x")
```

```
Loaded 581 words
pa used 7 x
ein used 6 x
so used 6 x
weg used 6 x
gerbers used 5 x
in used 5 x
da used 5 x
ja used 4 x
te used 4 x
wie used 4 x
```

Dictionary: shrnutí

- Rychlé operace vkládání a čtení, průměrně $\mathcal{O}(1)$, nejhorší případ $\mathcal{O}(n)$
- Nutnost implementace hash funkce
- Citlivé na výběr hash funkce a velikosti tabulky
- Nelze porovnávat velikost

Dictionary: in

Operátor in:

- Vrací True/False podle výskytu prvku v poli/stringu/dictionary
- Složitost $\mathcal{O}(n)$ pokud jsou data v poli nebo stringu
- Složitost $\mathcal{O}(1)$ pokud jsou data v dictionary

```
1 a = ["a", "b", 34, -1, 0]
2 d = { item:1 for item in a }
3 print(a)
4 print(d)
5 print(0 in a)   #O(n)
6 print(0 in d)   #O(1)
```

```
['a', 'b', 34, -1, 0]
{'a': 1, 'b': 1, 34: 1, -1: 1, 0: 1}
True
True
```

Dictionary: porovnání rychlosti vs pole


```
1  import timeit
2  fread = open("ewords.txt", "rt")
3  words = []
4  wordsDict = {}
5  for line in fread:
6      line = line.strip()
7      words.append(line)
8      wordsDict[ line ] = 1
9  fread.close()
10 print("Loaded", len(words), "words")
11 TRIALS = 1000*10
12 for word in ["zone", "positive", "factory", "nonexistingword!#$"]:
13     r = timeit.timeit(stmt="'{}' in words".format(word), globals=
           globals(), number=TRIALS)
14     print(word,(r / TRIALS)*1e6, "us/call in array")
15     r = timeit.timeit(stmt="'{}' in wordsDict".format(word),
           globals=globals(), number=TRIALS)
16     print(word,(r / TRIALS)*1e6, "us/call in dictionary")
```

- Slovník obsahuje seřazená slova (dle ang. abecedy)
- Hledáme slova ze začátku (“factory”), z prostředka (“positive”), z konce (“zone”) seznamu, a neexistující slovo

Dictionary: porovnání rychlosti vs pole


```
Loaded 3000 words
zone 13.472095795441419 us/call in array
zone 0.019873096607625484 us/call in dictionary
positive 8.72348679695278 4 us/call in array
positive 0.019883795175701 38 us/call in dictionary
factory 4.561067908070981 5 us/call in array
factory 0.019767892081290483 us/call in dictionary
nonexistingword!#$ 12.181531894020736 us/call in array
nonexistingword!#$ 0.016734190285205 84 us/call in dictionary
```

Hash funkce: použití

- Asociativní pole/dictionary — výpočet hodnoty (indexu) klíče
- Porovnání velkých dat — porovnává se pouze otisk (hash)
  - Například porovnání otisků prstů (spočítej hash otisku, porovnávej hash)
  - Kopírování/zálohování/synchronizace disků (např. `rsync`)
    - Program `md5sum`: výpočet hash pro soubory
    - Drobná změna v programu (bit) vede na změnu otisku
  - Použití hash místo originálních dat snižuje množství přenesených dat
- Prohledávání velkých stavových prostorů (šachy, go, . . .)
  - Pro zapamatování, které stavy byly navštíveny (ukládá se hash stavu, ne celý stav)

Dictionary: velikost tabulky

Dynamická realokace

- Pokud se tabulka zaplňuje ($\lambda > \lambda_{max}$) realokace, $m' \sim 2m$
- Pokud je $\lambda < \lambda_{min}$, realokace, $m' \sim m/2$
- Možné hodnoty: $m = 11$, $\lambda_{min} = 0.25$, $\lambda_{max} = 0.75$

Využití paměti: dictionary vs pole

- Dictionary

```
1 import psutil
2
3 a = {}
4 p = psutil.Process()
5 for n in range(100000):
6     a[n] = True
7     print(n, p.memory_info().rss)
```

- Pole

```
1 import psutil
2
3 a = []
4 p = psutil.Process()
5 for n in range(100000):
6     a.append(n)
7     print(n, p.memory_info().rss)
```

Využití paměti: dictionary vs pole