Rozptylová (Hash) tabulka

KIV/ADT – 5. přednáška

Miloslav Konopík

15. března 2024

Obsah

ADT Tabulka

ADT Tabulka – Opakování

- jednosměrný vztah mezi množinou klíčů a hodnot
- tabulka reprezentuje přiřazení jedné nebo žádné hodnoty každému klíči

Klíč

- diskrétní datový typ
  - `int`
  - `String`
  - ne `float`!

Nejdůležitější vlastnost

Musí být možné zjistit, zda jsou si dvě hodnoty rovny

Při vhodné implementaci vyhodnocení rovnosti je možné použít i složitější struktury (např. množina čísel $\{0, 1, 2\}$ je shodná s množinou $\{2, 0, 1\}$, množina celých čísel tedy může být klíčem)

Hodnota

- libovolný datový typ
- často reference na instanci třídy
  - instance obsahuje celou sadu dat
  - instance může být obsažena v několika různých datových strukturách (např. v Tabulce a zároveň v Seznamu)

Příklad (tabulka studentů)

Klíč: studijní číslo (`String`)
Hodnota: záznam studenta (reference na instanci třídy `Student`)

Klíč	Hodnota
A16N0123P	František Vonásek Křepelková 5 Hronov
A17N0321P	Kateřina Čumáčková Bezbřehá 8 Srnčí nad Smetanou
A17P0314P	Tomáš Marný Polévková 13 Želvice

Vlastnosti ADT Tabulka

Vlastnosti ADT Tabulka:

- Klíče jsou unikátní.
- Není určeno pořadí prvků.

Operace nad tabulkou (ideálně složitost operací (kromě získání všech klíčů): $\Theta(1)$):

- přidání přiřazení (klíč + hodnota),
- získání hodnoty pro daný klíč,
- zjištění, zda je klíči přiřazena hodnota,
- zrušení přiřazení (odebrání klíče),
- získání všech klíčů, kterým je přiřazena hodnota.

Vlastnosti ADT Tabulka:

- Klíče jsou unikátní.
- Není určeno pořadí prvků.

Operace nad tabulkou (ideálně složitost operací (kromě získání všech klíčů): $\Theta(1)$):

- přidání přiřazení (klíč + hodnota),
- získání hodnoty pro daný klíč,
- zjištění, zda je klíči přiřazena hodnota,
- zrušení přiřazení (odebrání klíče),
- získání všech klíčů, kterým je přiřazena hodnota.

Realizace tabulky

Tabulka realizovaná seznamem

- Datové prvky uložíme do ADT seznam.
- Pro realizaci dvojic použijeme `tuple`.

T12345P	Jakub
T45678P	Anna
T78901P	Martin
T10112P	Lucie
T11223P	Jan
T13141P	Petr
T14151P	Eva

Tabulka realizovaná seznamem – kód

```
1  data = [("T12345P", "Jakub"), ("T45678P", "Anna"), ("T78901P", "Martin"),
   ↪  ("T10112P", "Lucie"), ("T11223P", "Jan"), ("T13141P", "Petr"),
   ↪  ("T14151P", "Eva")]

2  code = "T78901P"                # Zvolíme data, která chceme najít

3  # Zkontrolujeme, zda jsou data v poli
4  for (c, n) in data:
5      if c == code:               # Porovnáme kód s daty
6          print(f"Jméno odpovídající {code} je {n}.")
7          break                   # Ukončíme cyklus
8  else:                           # Pokud jsme nenašli data
9      print(f"{code} není v poli dat.")
```

Tabulka realizovaná seznamem – kód

```
1  data = [("T12345P", "Jakub"), ("T45678P", "Anna"), ("T78901P", "Martin"),
   ↪  ("T10112P", "Lucie"), ("T11223P", "Jan"), ("T13141P", "Petr"),
   ↪  ("T14151P", "Eva")]

2  code = "T78901P"                # Zvolíme data, která chceme najít

3  # Zkontrolujeme, zda jsou data v poli
4  for (c, n) in data:
5      if c == code:               # Porovnáme kód s daty
6          print(f"Jméno odpovídající {code} je {n}.")
7          break                   # Ukončíme cyklus
8  else:                           # Pokud jsme nenašli data
9      print(f"{code} není v poli dat.")
```

$\Theta(n)$

Tabulka realizovaná seznamem a seřazená – kód

```
1  data = ...

2  data.sort(key=lambda tup: tup[0]) # Seřadíme data podle prvního prvku

3  code = "T78901P"                  # Zvolíme data, která chceme najít

4  # Zavoláme funkci bisect_left pro nalezení indexu kódu
5  index = bisect.bisect_left(data, (code,))

6  # Zkontrolujeme výsledek vyhledávání
7  if index != len(data) and data[index][0] == code:
8      print(f"Jméno odpovídající {code} je {data[index][1]}.")
9  else:
10     print(f"{code} není v poli dat.")
```

Tabulka realizovaná seznamem a seřazená – kód

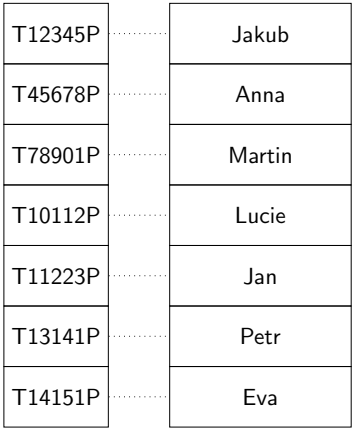
```
1  data = ...
2  data.sort(key=lambda tup: tup[0]) # Seřadíme data podle prvního prvku
3  code = "T78901P"                  # Zvolíme data, která chceme najít
4  # Zavoláme funkci bisect_left pro nalezení indexu kódu
5  index = bisect.bisect_left(data, (code,))
6  # Zkontrolujeme výsledek vyhledávání
7  if index != len(data) and data[index][0] == code:
8      print(f"Jméno odpovídající {code} je {data[index][1]}.")
9  else:
10     print(f"{code} není v poli dat.")
```

$\Theta(\log_2(n))$

Kvíz I

Tabulka realizovaná seznamy

- Datové prvky uložíme do dvou ADT seznam.
- Každému klíči na indexu i z pole klíčů odpovídají data na stejném indexu i z pole hodnot.

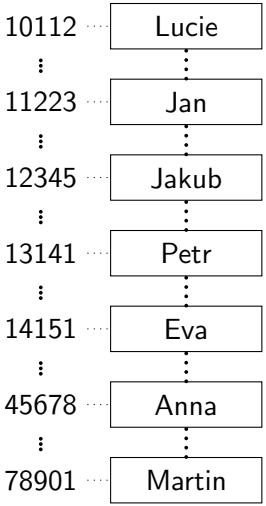


Tabulka realizovaná dvěma seznamy – kód

```
1  codes = ["T12345P", "T45678P", "T78901P", "T10112P", "T11223P",
   ↪  "T13141P", "T14151P"]
2  names = ["Jakub", "Anna", "Martin", "Lucie", "Jan", "Petr", "Eva"]

3  data = "T78901P"                # Zvolíme data, která chceme najít

4  for i in range(len(codes)):  # Projdeme všechny prvky v prvním poli
5      if codes[i] == data:       # Porovnáme prvek s daty
6          print(f"Jméno odpovídající {data} je {names[i]}.")
7          break                  # Ukončíme cyklus
8  else:                          # Pokud jsme nenašli data
9      print(f"{data} není v poli kódů.")
```

Tabulka s přímým adresováním

- Máme jen pole hodnot.
- Index do pole spočítáme přímo z klíče.



Tabulka s přímým adresováním – kód

```
1  data = [("T12345P", "Jakub"), ("T45678P", "Anna"), ("T78901P", "Martin"),
   ↪  ("T10112P", "Lucie"), ("T11223P", "Jan"), ("T13141P", "Petr")]

2  new_data = [None] * 100000        # Vytvoříme nové pole s číselnými klíči
3  for (key, name) in data:
4      key = key.replace("T", "").replace("P", "") # Odstraníme písmena z klíče
5      key = int(key)                              # Převedeme klíč na celé číslo
6      new_data[key] = name          # Přidáme jméno na index odpovídající klíči

7  key = T78901P                     # Zvolíme klíč, který chceme najít
8  key = key.replace("T", "").replace("P", "") # Odstraníme písmena z klíče

9  # Zkontrolujeme výsledek vyhledávání
10 if key < len(new_data) and new_data[key] is not None:
11     print(f"Jméno odpovídající {key} je {new_data[key]}.")
12 else:
13     print(f"{key} není v poli dat.")
```

Tabulka s přímým adresováním – kód

```
1  data = [("T12345P", "Jakub"), ("T45678P", "Anna"), ("T78901P", "Martin"),
   ↪  ("T10112P", "Lucie"), ("T11223P", "Jan"), ("T13141P", "Petr")]

2  new_data = [None] * 100000        # Vytvoříme nové pole s číselnými klíči
3  for (key, name) in data:
4      key = key.replace("T", "").replace("P", "") # Odstraníme písmena z klíče
5      key = int(key)                              # Převedeme klíč na celé číslo
6      new_data[key] = name          # Přidáme jméno na index odpovídající klíči

7  key = T78901P                     # Zvolíme klíč, který chceme najít
8  key = key.replace("T", "").replace("P", "") # Odstraníme písmena z klíče

9  # Zkontrolujeme výsledek vyhledávání
10 if key < len(new_data) and new_data[key] is not None:
11     print(f"Jméno odpovídající {key} je {new_data[key]}.")
12 else:
13     print(f"{key} není v poli dat.")
```

$\Theta(1)$

Rozptylová tabulka (Hash table)

Rozptylová tabulka (Hash table)

Základní myšlenky a vlastnosti:

- Seznam M přihrádek pro ukládání položek.
- Rozptylová funkce (hash function): mapuje klíč k na číslo přihrádky $i : 0 \ldots L$ ($k \in \mathbb{Z}_0^L$).
- Více položek v jedné přihrádce = kolize (collision/clash).
- Operace jsou rychlé, protože víme, v které přihrádce hledat.
- V každé přihrádce se snažíme držet omezený počet položek.

Implementace:

- Obdobná jako Tabulka s přímým adresováním.
- Kolize mohou být řešení dalšími vnořenými seznamy.

Zaplnění tabulky

Zaplnění tabulky (load factor):

- Průměrný počet položek na přihrádku.
- Load factor

$$\lambda = \frac{\text{počet položek } n}{\text{počet přihrádek } m}$$

.

- Velké $\lambda \rightarrow$ hodně kolizí $\rightarrow$ zpomalení operací
- Malé $\lambda \rightarrow$ hodně prázdných položek $\rightarrow$ nevyužitá paměť

Zaplnění tabulky

Zaplnění tabulky (load factor):

- Průměrný počet položek na přihrádku.
- Load factor

$$\lambda = \frac{\text{počet položek } n}{\text{počet přihrádek } m}$$

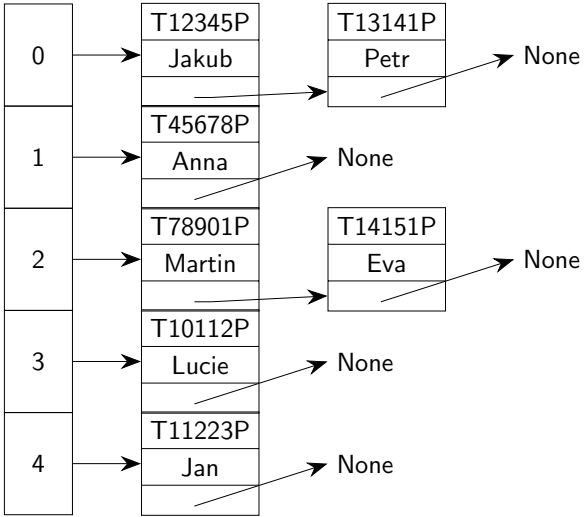
.

- Velké $\lambda \rightarrow$ hodně kolizí $\rightarrow$ zpomalení operací
- Malé $\lambda \rightarrow$ hodně prázdných položek $\rightarrow$ nevyužitá paměť

Zmenšování a zvětšování tabulky:

- Implementace sami zajišťují zvětšování a zmenšování tabulky na optimální λ.

Rozptylová tabulka – příklad

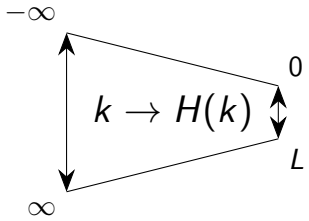


Rozptylová tabulka – kód

```
1  data = [("T12345P", "Jakub"), ("T45678P", "Anna"), ("T78901P", "Martin"),
   ↪  ("T10112P", "Lucie"), ("T11223P", "Jan"), ("T13141P", "Petr"),
   ↪  ("T14151P", "Eva")]  # Vytvoření seznamu n-tic

2  new_data = {}          # Vytvoření prázdného slovníku
3  for key, name in data:
4      new_data[key] = name  # Naplnění slovníku daty z n-tic

5  key = T78901P          # Nastavení klíče na hodnotu 78901

6  if key in new_data:    # Kontrola, zda se klíč nachází v slovníku
7      print(f"Jméno odpovídající {key} je {new_data[key]}.")
8  else:
9      print(f"{key} není v poli dat.")
```

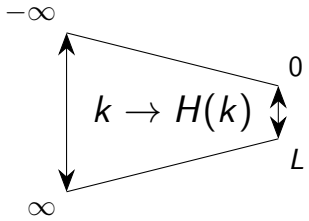
Rozptylová funkce

- Rozptylové (hash) funkce $i = H(k)$
  - vstup: klíč k
  - výstup: index v poli i
  - $i : 0 \ldots L$



Rozptylová funkce

- Rozptylové (hash) funkce $i = H(k)$
  - vstup: klíč k
  - výstup: index v poli i
  - $i : 0 \ldots L$

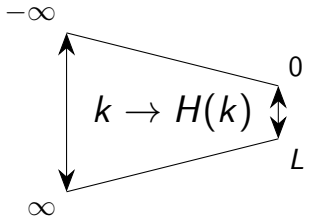


Vlastnosti rozptylové funkce:

- Náhodné hodnoty – rovnoměrné zaplňování seznamu přihrádek.
- Pro různé klíče by měla dávat různé hodnoty
  - nelze vždy, protože počet možných klíčů může být významně větší než délka pole
- Pro stejné hodnoty musí dávat stejné hodnoty
  - normalizace klíče.

Rozptylová funkce

- Rozptylové (hash) funkce $i = H(k)$
  - vstup: klíč k
  - výstup: index v poli i
  - $i : 0 \ldots L$



Vlastnosti rozptylové funkce:

- Náhodné hodnoty – rovnoměrné zaplňování seznamu přihrádek.
- Pro různé klíče by měla dávat různé hodnoty
  - nelze vždy, protože počet možných klíčů může být významně větší než délka pole
- Pro stejné hodnoty musí dávat stejné hodnoty
  - normalizace klíče.

Rovnost hodnoty rozptylové funkce pro různé klíče: kolize

Normalizace klíče

Nejčastěji normalizujeme řetězce:

- Převod na stejnou velikost a ostranění mezer: `str_key.lower().strip()`.
- Odstranění diakritiky: `from unidecode import unidecode; unidecode(str_key)`
- Odstranění bílých znaků.

Další normalizované datové typy:

- Datum a čas.
- Měna.
- Odstranění bílých znaků.

Rozptylová funkce v Pythonu

Vestavěná funkce `hash()`

- Je definována pro neměnitelné (immutable) datové typy: čísla, řetězce, n-tice (tuple), neměnné množiny (frozenset), objekty.
- Nelze používat pro seznamy, tabulky, množiny...

```
1  hash(123)                                # 123
2  hash(12345678912345678912345678 9)       # 1361126810988140292

3  hash("python")                           # -5859315622727106325

4  hash(("Petr", "p@a.cz"))                 # 2812264813959488621

5  hash({1, 2, 3})                          # TypeError: unhashable type: 'set'
6  hash(frozenset({1, 2, 3}))               # -272375401224217160
```

Rozptylová funkce pro vlastní objekty – špatně

```
1  class User:
2  # Inicializujte uživatele s jménem a e-mailem
3  def __init__ (self, name, email):
4      self.name = name
5      self.email = email

6  hash(User("Petr", "p@a.cz"))                          # 270880913
7  hash(User("Petr", "p@a.cz"))                          # 270880919
8  User("Petr", "p@a.cz") == User("Petr", "p@a.cz")   # False
```

Rozptylová funkce pro vlastní objekty – špatně

```
1  class User:
2  # Inicializujte uživatele s jménem a e-mailem
3  def __init__ (self, name, email):
4      self.name = name
5      self.email = email

6  hash(User("Petr", "p@a.cz"))                          # 270880913
7  hash(User("Petr", "p@a.cz"))                          # 270880919
8  User("Petr", "p@a.cz") == User("Petr", "p@a.cz")      # False
```

Pro vlastní objekty je nutné implementovat funkce

- `__eq__(self, other)`
- `__hash__(self)`

Rozptylová funkce pro vlastní objekty – správně

```
1       def __eq__(self, value: object) -> bool:
2           if not isinstance(value, User):
3               return False

4           return self.name == value.name and self.email == value.email

5       def __hash__(self) -> int:
6           return hash((self.name, self.email))

7   hash(User("Petr","p@a.cz"))                    # 2061414140790974 67
8   hash(User("Petr","p@a.cz"))                    # 2061414140790974 67
9   User("Petr","p@a.cz") == User("Petr","p@a.cz") # True
```

Rozptylová funkce pro vlastní objekty – Poznámka

Implementace vlastní `__hash__(self)` funkce

Implementace vlastní `__hash__(self)` funkce nezohledňuje velikost tabulky. Vracíme velké číslo, které si následně ADT tabulka transformuje do potřebného rozsahu.

Kvíz II

Použití rozptylové funkce v tabulce

Vložení hodnoty pro klíč k

- hodnota se vloží na index $H(k)$,
- pokud je index $H(k)$ již obsazen, tak se hodnota přidá do seznamu kolizí.

Použití rozptylové funkce v tabulce

Vložení hodnoty pro klíč k

- hodnota se vloží na index $H(k)$,
- pokud je index $H(k)$ již obsazen, tak se hodnota přidá do seznamu kolizí.

Získání hodnoty pro klíč k

- prohledává se seznam kolizí na indexu H(k).

Použití rozptylové funkce v tabulce

Vložení hodnoty pro klíč k

- hodnota se vloží na index $H(k)$,
- pokud je index $H(k)$ již obsazen, tak se hodnota přidá do seznamu kolizí.

Získání hodnoty pro klíč k

- prohledává se seznam kolizí na indexu H(k).

Zrušení přiřazení klíče k

- prohledává se seznam kolizí na indexu H(k),
- pokud nalezen shodný klíč, je ze seznamu vyjmut.

Složitosti operací s rozptylovou tabulkou

Očekávané (průměrné):

- Přidání přiřazení (klíč + hodnota): $\Theta(1)$.
- Získání hodnoty pro daný klíč: $\Theta(1)$.
- Zjištění, zda je klíči přiřazena hodnota: $\Theta(1)$.
- Zrušení přiřazení (odebrání klíče): $\Theta(1)$.

Nejhorší případ:

- Přidání přiřazení (klíč + hodnota): $\Theta(n)$.
- Získání hodnoty pro daný klíč: $\Theta(n)$.
- Zjištění, zda je klíči přiřazena hodnota: $\Theta(n)$.
- Zrušení přiřazení (odebrání klíče): $\Theta(n)$.

Implementace ADT množina

Implementace ADT množina

Můžeme implementovat stejně jako ADT Tabulka.

- Ve struktuře uchováváme pouze klíče.
- Vše ostatní je stejné.

References I