Hashovanie

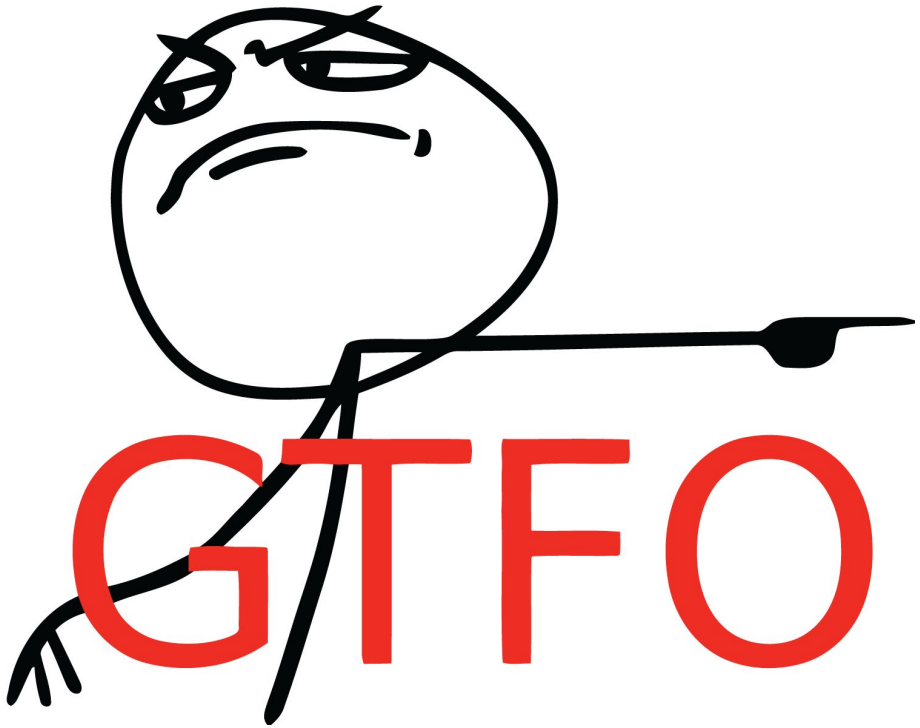
Adam Valach

FIIT STU

2021/22

Opakovanie – čo je to heš

- Citoslovce odháňania
- V spoločnosti často nezdvorilé



Mali sme stromy

- AVL:

Algorithm	Average	Worst case
Search	$\Theta(\log n)$[1]	$O(\log n)$[1]
Insert	$\Theta(\log n)$[1]	$O(\log n)$[1]
Delete	$\Theta(\log n)$[1]	$O(\log n)$[1]

- Splay:

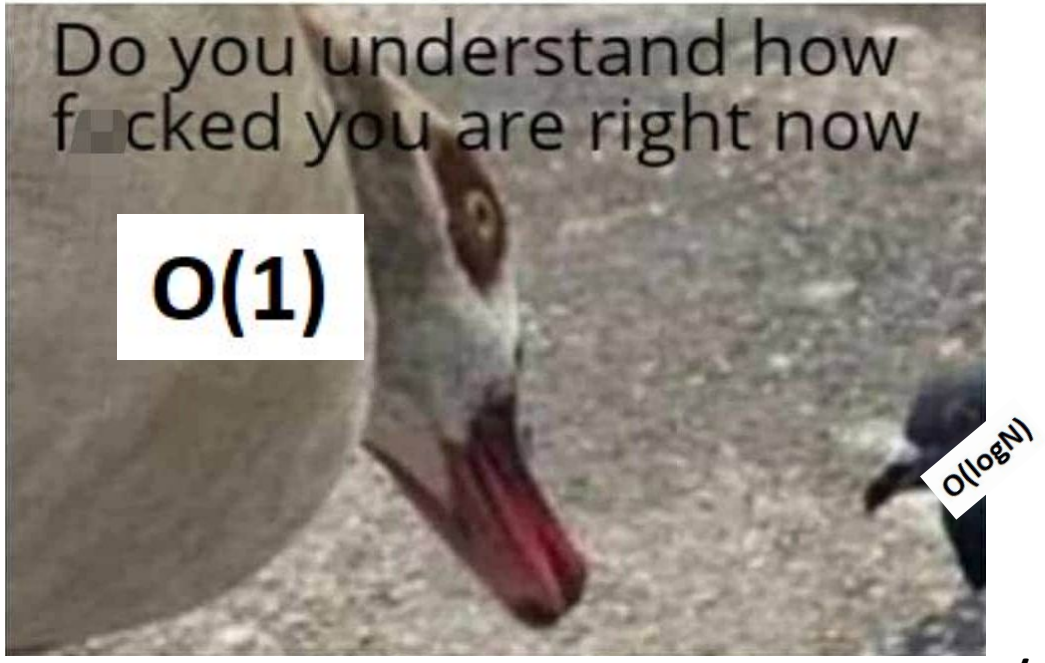
Function	Amortized	Worst Case
Search	$O(\log n)$[1]:659	$O(n)$[2]:1
Insert	$O(\log n)$[1]:659	$O(n)$
Delete	$O(\log n)$[1]:659	$O(n)$

Cool 👌

- Nakoľko boli zoradené podľa nejakých pravidiel, nemuseli sme toľko prehľadávať 👌

How about O(1)?

- Čo keby sme však vedeli povedať, kde presne sa prvok nachádza?

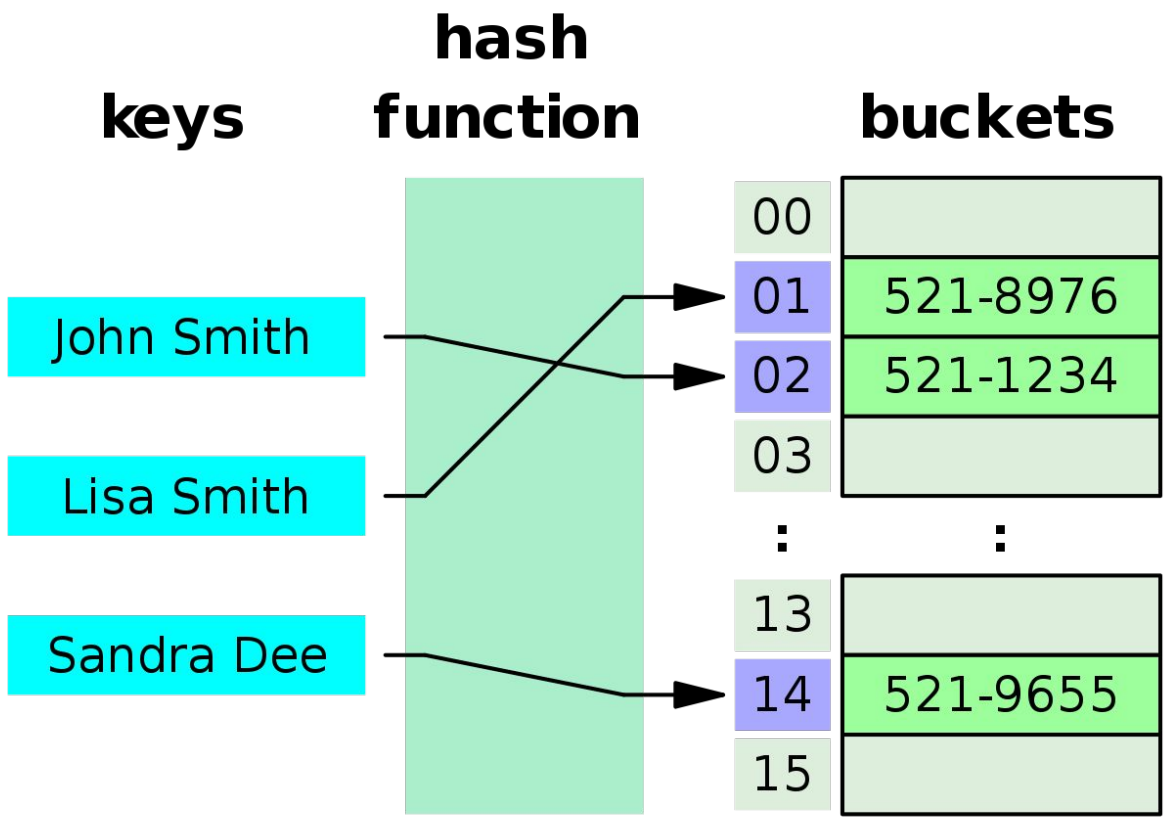


Hashovacia tabuľka

- Dátová štruktúra 😮
- Sú v nej asociované kľúče s hodnotami („key-value pair“)

Algorithm	Average	Worst case
Search	O(1)	O(*n*)
Insert	O(1)	O(*n*)
Delete	O(1)	O(*n*)

Príklad s 📞#



Na základe výstupu hashovacej funkcie vieme zatriediť daný pár

Laický pohľad

- Máme nejakú tabuľku o nejakej veľkosti
- Máme dvojicu kľúč:hodnota
- Do hashovacej funkcie hodíme kľúč
- Výstup funkcie bude číselná hodnota
- *(výstup_funkcie mod veľkosť_tabuľky)* nám dá index
- GGWP

```
typedef struct node {
    int key;
    char value[420];
} NODE;
```

Hashovacia funkcia

- Funkcia pre prevod vstupného reťazca dát na krátky výstupný reťazec*
- Čo tým chce autor povedať je, že funkcia dostane vstup a na výstupe vráti spracovaný vstup v nejakej fixnej veľkosti a forme (vráti integer/32 bajtový hex string...)
- Parametre dobrej hashovacej funkcie
  - Rýchly výpočet
  - Minimálny počet kolízii – Rovnomerne rozhádže prvky do tabuľky

*Wikipedia

Príklad

- Hashovacia funkcia berie na vstupe reťazec a vracia súčet všetkých jeho znakov
- #1: Kľúč: ***„Yeet“***
  - 89+101+101+116 = **407**
- #2: Kľúč: ***„Ligma“***
  - 76+105+103+109+97 = **490**
- #3: Kľúč: ***„Langos s kecupom“***
  - 76+97+110+103+111+115+32+115+32 +107+101+99+117+112+111+109 = **1547**

```
int hashFunction(char input[]) {
    int output = 0;
    for(int i = 0; i < strlen(input); i++) {
        output += (int)(input[i]);
    }
    return output;
}
```

Identifikácia miesta na ukladanie

- Index identifikujeme pomocou: *h(x) % veľkosť_tabulky*
- Príkad:
  - Uvažujme hashovaciu tabuľku o veľkosti 1337
  - Majme produkty a ich ceny, ktoré chceme uschovať v hashovacej tabuľke pre rýchly prístup k dátam a pomalšie štruktúry nestačia, lebo miestna Karen chce byť obslúžená rýchlo.
  - „Langos s kecupom“ má po zahashovaní hodnotu 1547
  - V hashovacej tabuľke bude uložený na miesto 1547 % 1337 = 210
  - Keď budeme chcieť k nemu a jeho cene neskôr pristúpiť (search), prvé miesto kam sa pozrieme bude index 210

KEY	Langos s kecupom
VALUE	3,90 €

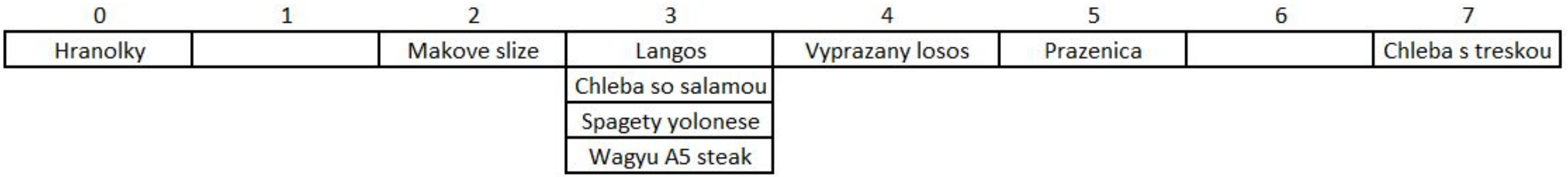
Kolízia

- Nastane, keď prvky majú rôzne kľúče ale rovnaký výsledok hashovacej funkcie
- ***h(key1) == h(key2)***
- Teda uplatnením ***h(key) % veľkosť_tabuľky*** by sme dostali rovnaký index (čo je big no-no)

Riešenie kolízii

Zreťazenie

- Umožníme aby na jednom mieste mohlo byť viac prvkov (spájaný zoznam)
- Očakávaný čas na vyhľadanie prvku v takejto tabuľke je O(α)
- α je tzv. faktor naplnenia – N/M
  - N – počet prvkov v tabuľke
  - M – veľkosť tabuľky



Otvorené adresovanie

- Na jednom mieste môže byť maximálne jeden prvok
- V prípade kolízie uložíme prvok na najbližšie voľné miesto v tabuľke
- V prípade, že tabuľka dosiahne definovaný limit α (faktor naplnenia), tak ju zväčšíme a prvky nanovo prehashujeme
  - ***h(key) % size*** bude teraz vracať iný index (lebo sa zmenil ***size***)

Otvorené adresovanie - Insert

- Ideme pridať prvok „Spagety yolonese“ do tabuľky, rovnako ako „Langos“ má index ***3***
- Ak je okienko ***h(key) % size*** plné, skúsime ***(h(key) + 1) % size,*** ak aj to ***(h(key) + 2) % size***... až kým nenájdeme voľné miesto
- Prvok „Spagety yolonese“ nepôjde do okienka ***3***, ale ***4***

0	1	2	3	4	5	6	7
Hranolky		Makove slize	Langos				

0	1	2	3	4	5	6	7
Hranolky (0)		Makove slize (2)	Langos (3)	Spagety yolonese (3)			

Otvorené adresovanie - Search

- Keď chceme nájsť prvok „Spagety yolonese“ z minulého príkladu, pozrieme sa najprv na miesto ***h(„Spagety yolonese“) % size = 3***
- Na tomto mieste sa nenachádza hľadaný prvok, pozrieme sa na miesto ***h(„Spagety yolonese“) + 1) % size***
- Na tomto mieste sme našli kľúč „Spagety yolonese“ vrátime hodnotu

0	1	2	3	4	5	6	7
Hranolky (0)		Makove slize (2)	Langos (3)	Spagety yolonese (3)			

Otvorené adresovanie - Delete

- Chceme vymazať prvok „Langos“
- Nájdeme ho na prvýkrát na mieste 3
- Nemôžeme ho však nechať prázdne, lebo neskôr pri vyhľadávaní prvku „Spagety yolonese“, by sme na mieste 3 videli prázdne miesto a ďalej už nehľadali

0	1	2	3	4	5	6	7
Hranolky (0)		Makove slize (2)	Langos (3)	Spagety yolonese (3)			

0	1	2	3	4	5	6	7
Hranolky (0)		Makove slize (2)	DELETED	Spagety yolonese (3)			

Otvorené adresovanie – Search #2

- Keď chceme nájsť prvok „Spagety yolonese“ z minulého príkladu, pozrieme sa najprv na miesto ***h(„Spagety yolognese“) % size = 3***
- Na tomto mieste je informácia, že bolo odtiaľ vymazávané, pozrieme sa na miesto ***h(„Spagety yolognese“) + 1) % size***
- Na tomto mieste sme našli kľúč „Spagety yolognese“ vrátime hodnotu

0	1	2	3	4	5	6	7
Hranolky (0)		Makove slize (2)	DELETED	Spagety yolonese (3)			

Otvorené adresovanie – Search #3

- Keď chceme nájsť v tabuľke prvok s kľúčom „Chleba so salamou“
  - h(„Chleba so salamou “) = 3
- Toto miesto je prázdne, ale je v ňom informácia, že bol odtiaľ vymazaný prvok, preto hľadáme ďalej
- Na indexe 4 sme našli iný kľúč, hľadáme ďalej
- Index 5 je prázdny, search končí, vieme, že prvok sa tam nenachádza

0	1	2	3	4	5	6	7
Hranolky (0)		Makove slize (2)	DELETED	Spagety yolonese (3)			