Typografie a publikování – 5. projekt

Hashovacia tabuľka

Jakub Bartko
xbartk07@stud.fit.vutbr.cz

Vysoké učení technické v Brně
Fakulta informačních technologií

26. apríla 2021

Motivácia a základy

Hashovacia tabuľka je jednou zo základných abstraktných dátových štruktúr

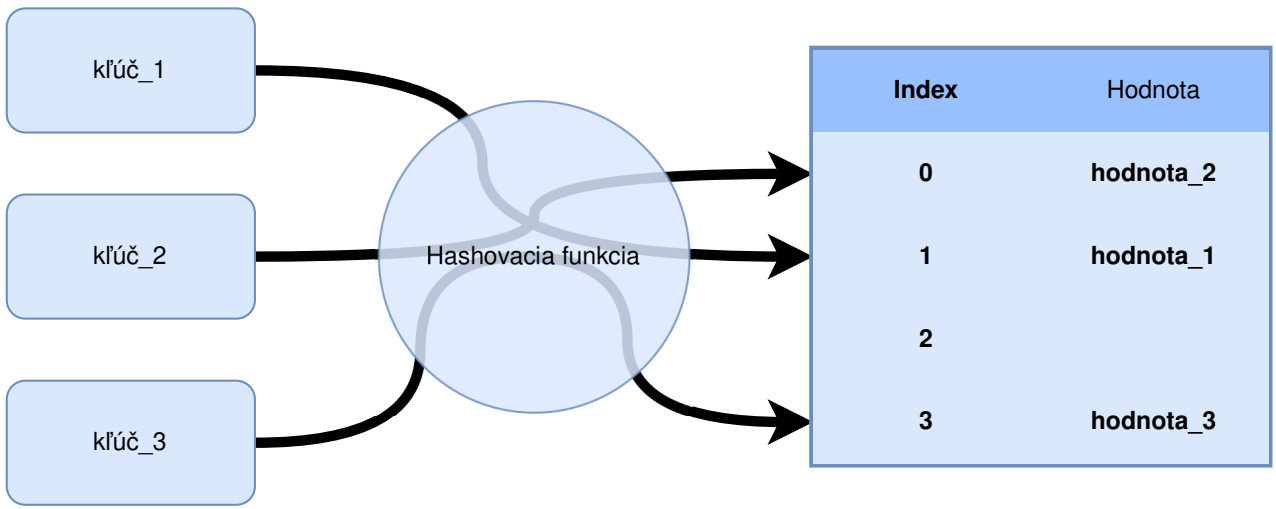
- asociatívne pole využívajúce **hashovaciu funkciu** na mapovanie kľúčov na hodnoty
- **hashovacia funkcia** pre zadaný kľúč vypočíta *hash*, ktorý určuje konkrétny *index*, na ktorom sa hľadaná hodna nachádza

Prečo práve hashovacia tabuľka, oproti obyčajnému poľu alebo jednosmerne viazanému zoznamu?

- zvýšenie efektivity najpoužívanejších operácií
- lineárne prehľadávanie prvkov je v hashovacej tabuľke poslednou možnosťou, no v zozname samozrejmosťou

Ďalšie vlastnosti

- Priemerná zložitosť hashovacej tabuľky je *O(1)*, najhoršia zložitosť je *O(n)* (degraduje na lineárny zoznam),
- veľkosť vyberajte tak, aby *zaplnenie nepresiahlo 70%*,
- voľba prvočísla ďalej znižuje výskyt kolízií indexov.



Obr. 1: Princíp hashovacej tabuľky

Výber hashovacej funkcie

- Využitím hashovania získavame rýchlosť vyhľadávania na úkor využívaného dátového priestoru.
- Najbežnejšie metódy hashovania: *(podľa zložitosti)*
  1. bitové operácie,
  2. násobenie,
  3. delenie.
- Okrem komplexnosti sa zohľadňuje najmä *rovnomerná distribúcia odlišných kľúčov* medzi indexy.
- Príkladmi efektívnych hashovacích funkcií sú *MurmurHash*[1] alebo *Fowler–Noll–Vo*[2]

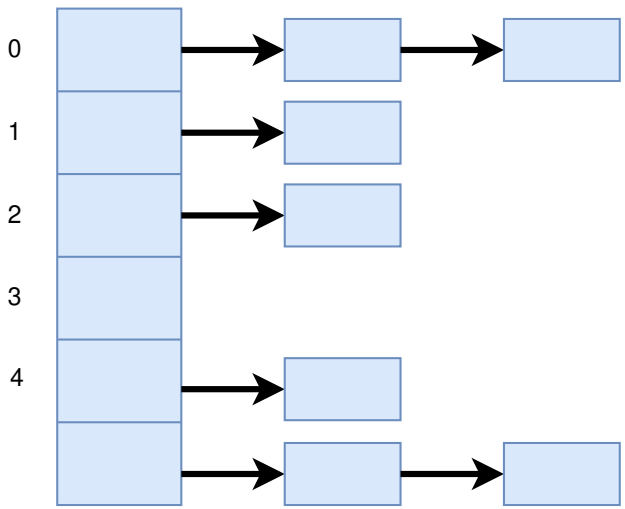
[1] https://en.wikipedia.org/wiki/MurmurHash

[2] https://w.wiki/XKZ

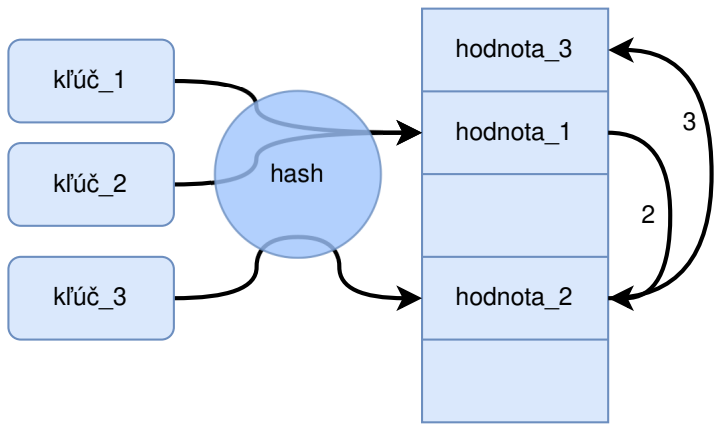
Riešenie synoným

*Čo s rôznymi kľúčmi, ktoré sa namapujú na rovnaký index (**synonymá**)?*

- **Explicitné** zreťazenie – napr. linerárny zoznam na každom indexe
- **Implicitné** zreťazenie – cieľový index získaný z pôvodného indexu



(a) Explicitné zreťazenie s lineárnym zoznamom



(b) Implicitné zreťazenie s pevným krokom 2

Search

Princíp: Na príslušnom indexe prejdi zoznam prvkov a vráť ten s hľadaným kľúčom

Algoritmus 0: Vyhľadávanie prvku podľa kľúča `KEY`

```
index = hashFunc(KEY);
bucket = hashTable[index];
item = bucket.start;
while (item is not bucket.end) do
    if (item.key is KEY) then
        return item;
    else
        item = item.next;
    end
end
return NOT_FOUND;
```

Insert

Princíp: Na príslušnom indexe nájdi voľnú pozíciu a vlôž na ňu nový prvok

Algoritmus 1: Vkladanie prvku s kľúčom `KEY`

```
index = hashFunc(new_item.KEY);
bucket = hashTable[index];
item = bucket.start;
while (item is not bucket.end) do
    if item is empty then
        item = new_item;
        return;
    else
        item = item.next;
    end
end
```

Delete

Princíp: Na príslušnom indexe nájdi hľadaný prvok a odstráň ho zo zoznamu

Algoritmus 2: Vymazanie prvku s kľúčom `KEY`

```
index = hashFunc(new_item.KEY);
bucket = hashTable[index];
item = bucket.start;
while (item is not bucket.end) do
    if (item.key is KEY) then
        delete item;
        link item.previous and item.next;
        return;
    else
        item = item.next;
    end
end
return NOT_FOUND;
```

Použité zdroje

 Predmet IAL – 6. prednáška ”Vyhledávací tabulky”, FIT VUT, 2020

 Which hashing algorithm is best for uniqueness and speed? `https://softwareengineering.stackexchange.com/qu`