

Algoritmi i strukture podataka

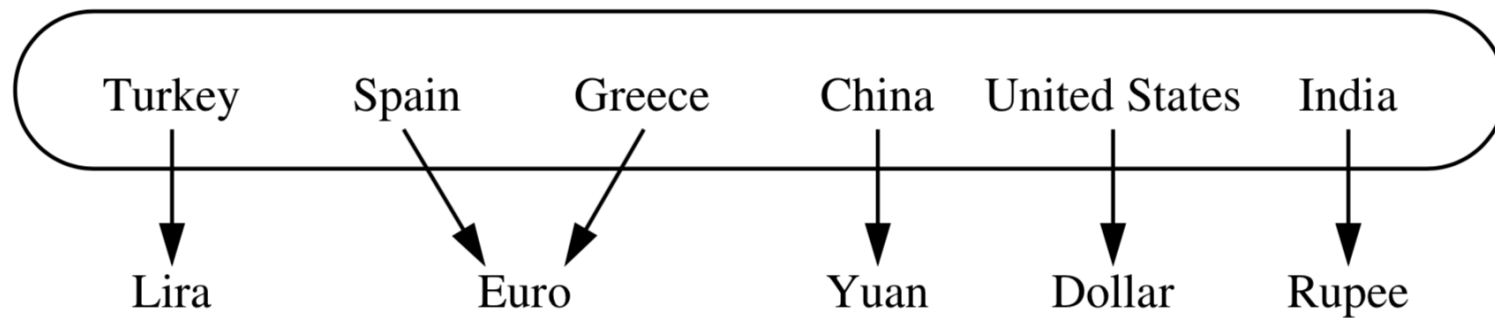
08 Hash map

Katedra za informatiku, Fakultet tehničkih nauka, Novi Sad

2023

Opšte informacije

- Element se sastoji od ključa i vrednosti.
- ključevi su jedinstveni (nema ponavljanja)
- vrednosti ne moraju biti jedinstvene



Operacije

<code>M[k]</code>	vraća vrednost v vezanu za ključ k u mapi M ; ako ne postoji, izaziva <code>KeyError</code> ; implementira je <code>__getitem__</code>
<code>M[k] = v</code>	dodeljuje vrednost v ključu k u mapi M ; ako ključ već postoji, zamenjuje staru vrednost; implementira je <code>__setitem__</code>
<code>del M[k]</code>	uklanja element sa ključem k iz mape M ; ako ne postoji, izaziva <code>KeyError</code> ; implementira je <code>_delitem__</code>
<code>len(M)</code>	vraća broj elemenata u mapi M ; implementira je <code>__len__</code>
<code>iter(M)</code>	generiše listu ključeva iz mape M ; implementira je <code>__iter__</code>

Operacije

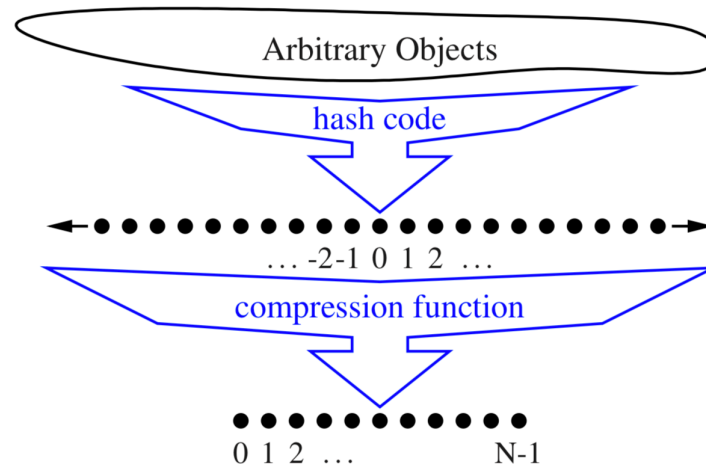
<code>k in M</code>	vraća True ako mapa M sadrži ključ k ; implementira je <code>__contains__</code>
<code>M.keys()</code>	vraća skup svih ključeva iz M
<code>M.values()</code>	vraća skup svih vrednosti iz M
<code>M.items()</code>	vraća skup svih parova (k,v) iz M

Hash funkcija

- Da bismo ubrzali pristup, mapiramo ključeve na memorijske lokacije
- Hash funkcija je svaka funkcija koja se koristi za mapiranje podataka određene dužine na podatke fiksne dužine
- Osobine:
 - Determinističnost – za iste ulaze dobijamo uvek iste izlaze
 - Uniformnost – ulazne vrednosti bi trebalo da se mapiraju na izlazni opseg na što uniformniji način. Svaka hash vrednost bi trebalo da se dobija sa približno istom verovatnoćom
 - Fiksni opseg
 - Ne postojanje inverzne funkcije

Hash funkcija

- $\text{hash}(\text{key}) = \text{compress}(\text{hash_code}(\text{key})) = \text{position}$
 - često se hash funkcija može posmatrati kao kompozicija dve funkcije:
 - **hash code**: mapira ključ na ceo broj
 - **compression function**: mapira hash kôd na broj u intervalu $[0, N - 1]$



Duplikati

- Prilikom heširanja ključa, postoji mogućnost da dva ili više ključeva dobije istu poziciju. Ukoliko se to desi, nastaje kolizija.
- U primeru sa predavanja, ukoliko se za heširanje koristi funkcija

$$\text{hash}(x) = x \bmod 10$$

- heširajući ključeve 17 i 107 dobijamo istu poziciju 7.

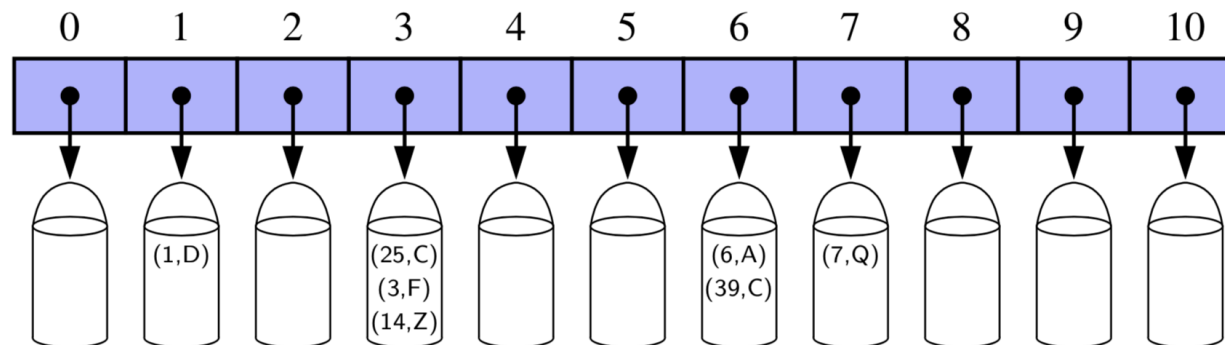
$$17 \bmod 10 = 7$$

$$107 \bmod 10 = 7$$

- Takve ključeve nazivamo duplikatima.

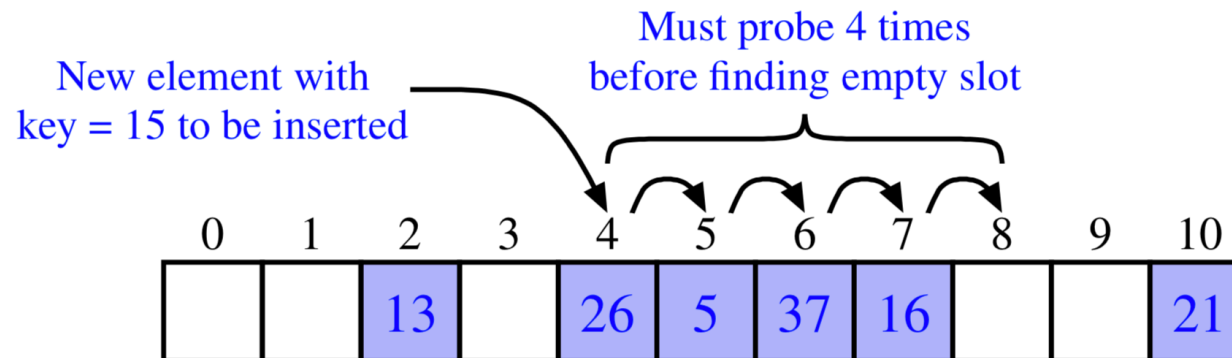
Bucket array

- pretvorićemo ključeve u cele brojeve pomoću **hash funkcije**
- dobra hash funkcija će ravnomerno distribuirati ključeve u $[0, N - 1]$
- ali može biti duplikata
- duplikate ćemo čuvati u „kantama“ – tzv. **bucket array**



Linearno traženje/smeštanje prekoračilaca

- **linear probing**: smešta element u koliziji u prvu sledeću slobodnu ćeliju (cirkularno)
- elementi u koliziji se nagomilavaju izazivajući dalje kolizije
- primer:



Linearno traženje - prekoračioc

- Sve ključeve koje nije bilo moguće smestiti na poziciju koja im priprada heširanjem nazivamo prekoračiocima.

Linearno traženje - Primer

- Uzmimo da koristimo heš funkciju
 $\text{hash}(\text{key}) = \text{key} \bmod 10$

Linearno traženje - Primer

- Dodajemo novi element sa ključem 19
- $19 \bmod 10 = 9$

0	1	2	3	4	5	6	7	8	9
									19

Linearno traženje - Primer

- Dodajemo novi element sa ključem 77
- $77 \bmod 10 = 7$

0	1	2	3	4	5	6	7	8	9
							77		19

Linearno traženje - Primer

- Želimo da pristupimo elementu sa ključem 58
- $58 \bmod 10 = 8$

0	1	2	3	4	5	6	7	8	9
							77		19

- Pozicija 8 je prazna
- Da li smemo da zaključimo da elementa sa ključem 58 nema?

Linearno traženje - Primer

- Dodajemo novi element sa ključem 37
- $37 \bmod 10 = 7$

0	1	2	3	4	5	6	7	8	9
							77	37	19

- Prekoračilac tj. duplikat
- Smeštamo ga prvu slobodnu lokaciju

Linearno traženje - Primer

- Želimo opet da pristupimo elementu sa ključem 58
- $58 \bmod 10 = 8$

0	1	2	3	4	5	6	7	8	9
							77	37	19

- Lokacija 8 je sada zauzeta
- Tražimo element na sledećoj lokaciji
- Kada postupak treba da se zaustavi?

Linearno traženje - Primer

- Dodajemo element sa ključem 58
- $58 \bmod 10 = 8$

0 1 2 3 4 5 6 7 8 9

58							77	37	19
-----------	--	--	--	--	--	--	-----------	-----------	-----------

- Lokacija 8 je sada zauzeta
- Element postaje prekoračilac
- Prva slobodna pozicija na 0 (cirkularno pristupamo lokacijama)

Linearno traženje - Primer

- Uklanjamo element sa ključem 37

0	1	2	3	4	5	6	7	8	9
58							77		19

Linearno traženje - Primer

- Želimo opet da pristupimo elementu sa ključem 58
- $58 \bmod 10 = 8$

0	1	2	3	4	5	6	7	8	9
58							77		19

- Nailazak na praznu lokaciju 8 može da nas navede na zaključak da elementa sa ključem 58 nema
- Sa slike vidimo da pretraga ne treba da se prekine na ovoj poziciji
- Kako da rešimo ovaj problem?

Linearno traženje - Primer

- Da smo znali da je na poziciji 8 bio smešten neki element pa da je posle izbrisan, mogli bismo da zaključimo da pretraga treba da se nastavi.
- Dakle, moramo razlikovati situaciju u kojoj na poziciji nikad nije bio upisan podatak od sitacije u kojoj je podatak bio upisan pa je naknadno izbrisan.
- Takvu poziciju označavamo sa X

0	1	2	3	4	5	6	7	8	9
58							77	X	19

- Nailazak na poziciju označenu sa X sugeriše da se potraga nastavlja

Linearno traženje - Primer

- Da zaključimo, pozicija može imati 3 stanja:
 - Slobodna (nikad nije bila zauzimana)
 - Zauzeta
 - Oslobođenja (bila je zauzeta, ali je element izbrisan)
- Kako bismo u kodu implementirali ova tri stanja?

Zadatak 1

- Implementirati klasu **HashMap** sa baketima.

Zadatak 2

- Implementirati klasu **HashMap** linearnim traženjem/smeštanjem prekoračilaca.