Algoritmi i strukture podataka

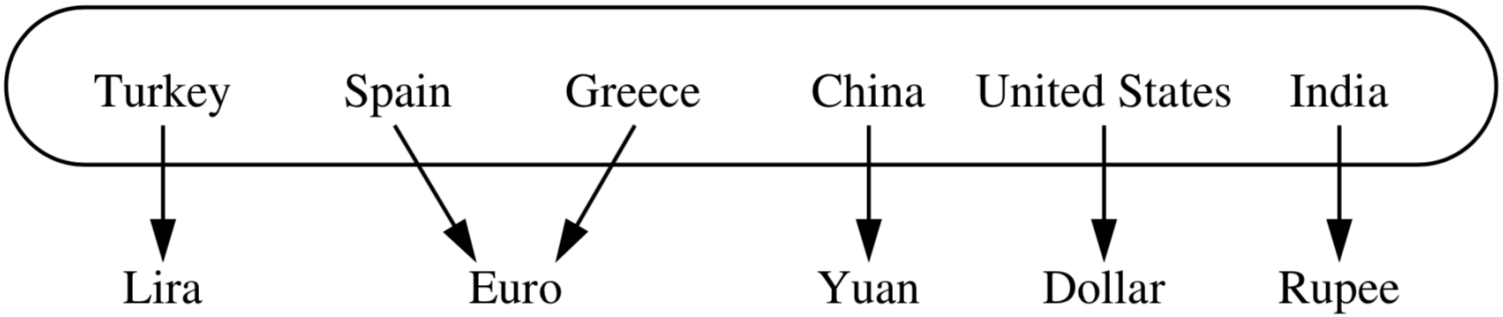
08 Hash map

Katedra za informatiku, Fakultet tehničkih nauka, Novi Sad

2024

Opšte informacije

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



Operacije

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

Operacije

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

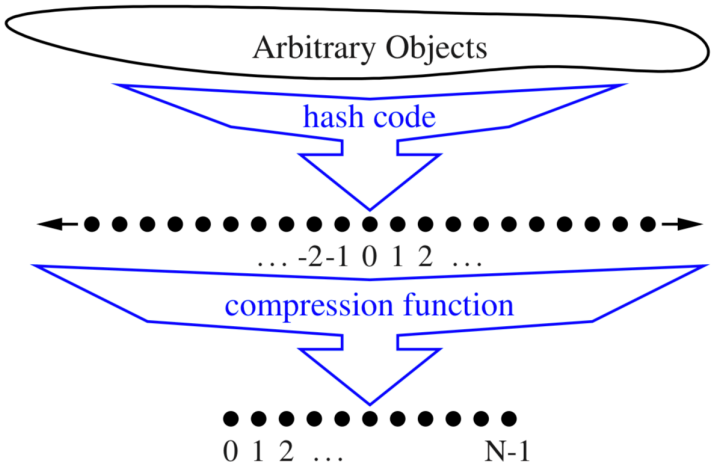
Hash funkcija

- 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
  - Fiksan opseg
  - Ne postojanje inverzne funkcije

Hash funkcija

- hash(key) = compress(hash_code(key)) = 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

```
hash(x) = x mod 10
```

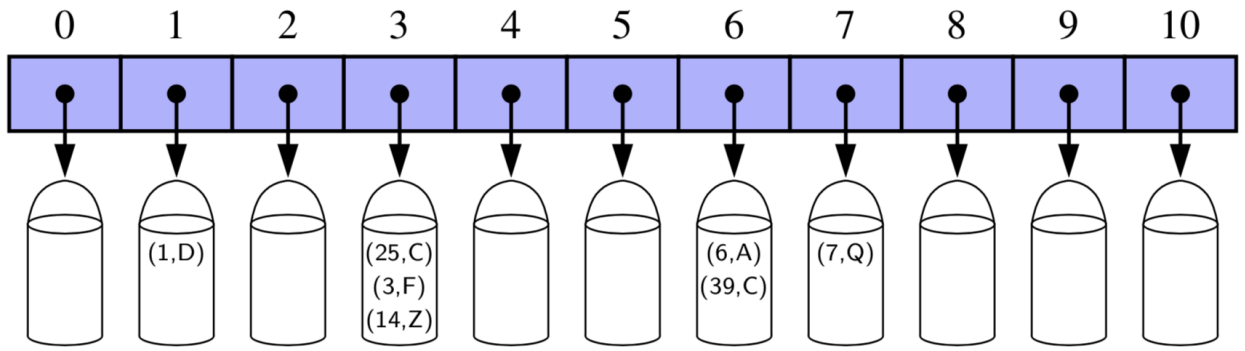
- heširajući ključeve 17 i 107 dobijamo istu poziciju 7.

```
17 mod 10 = 7
107 mod 10 = 7
```


- Takve ključeve nazivamo duplikatima.

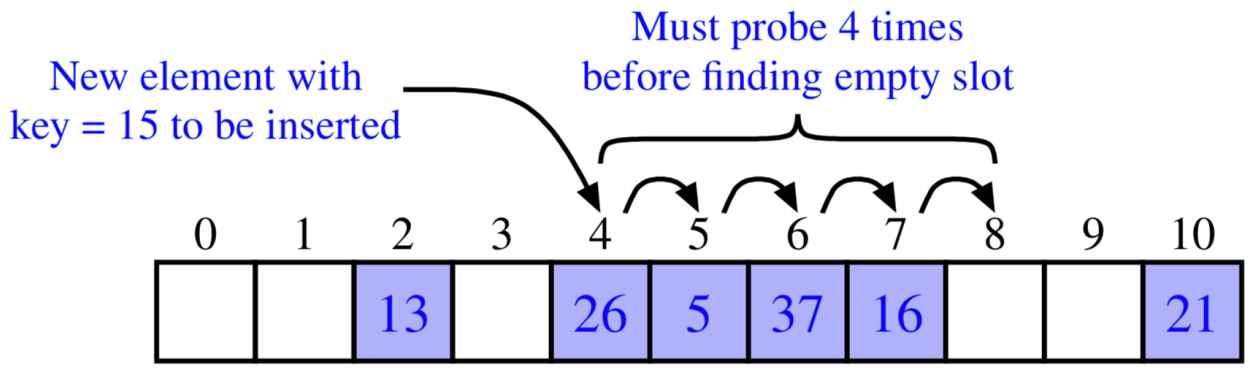
Bucket array

- 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 - prekoračioci

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

Linearno traženje - Primer

- Uzmimo da koristimo heš funkciju

```
hash(key) = key mod 10
```

Linearno traženje - Primer

- Dodajemo novi element sa ključem 19
- `19 mod 10 = 9`

Linearno traženje - Primer

- Dodajemo novi element sa ključem 77
- `77 mod 10 = 7`

Linearno traženje - Primer

- Želimo da pristupimo elementu sa ključem 58
- `58 mod 10 = 8`

- 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 mod 10 = 7`

- Prekoračilac tj. duplikat
- Smeštamo ga prvu slobodnu lokaciju

Linearno traženje - Primer

- Želimo opet da pristupimo elementu sa ključem 58
- `58 mod 10 = 8`

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

Linearno traženje - Primer

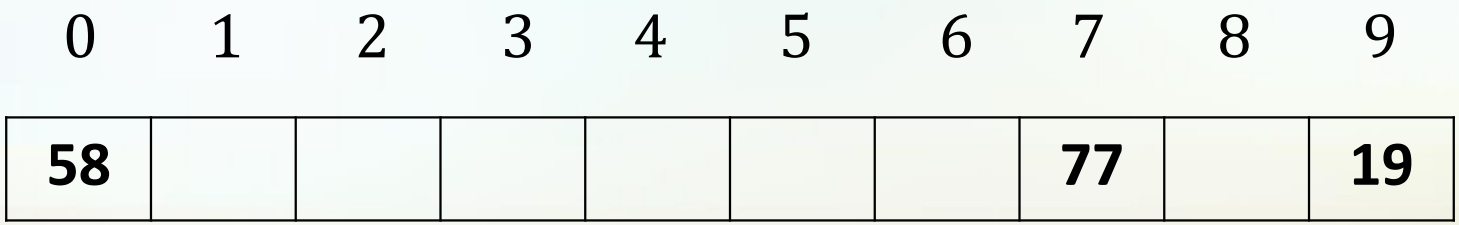
- Dodajemo element sa ključem 58
- `58 mod 10 = 8`

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



Linearno traženje - Primer

- Želimo opet da pristupimo elementu sa ključem 58
- `58 mod 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.