Algoritmi

HEŠIRANJE

“Rečnik” podataka (*dictionary*)

- apstraktan tip podataka (ADT – *Abstract Data Type*)
- sadrži (i održava) skup elemenata gde svaki ima ključ (*key*)
  - elemenat (*item*) se može posmatrati kao uređeni par {ključ, vrednost}
- osnovne operacije:
  - ubaci elemenat u skup: DODAJ(*el*)
  - ukloni elemenat iz skupa: OBRIŠI(*el*)
  - pronađi elemenat – ako postoji: PRETRAŽI(*ključ*) // traži identičan ključ
- poznat i kao „mapa“ – mapira ključ na vrednost
- **Koristi heširanje**
- Složenost operacija: $O(1)$ (u proseku po operaciji)
- Poređenje: npr. BSP ima složenost $O(\log n)$
  - u traženju, BSP može da ponudi naredni veći (ili manji) elemenat

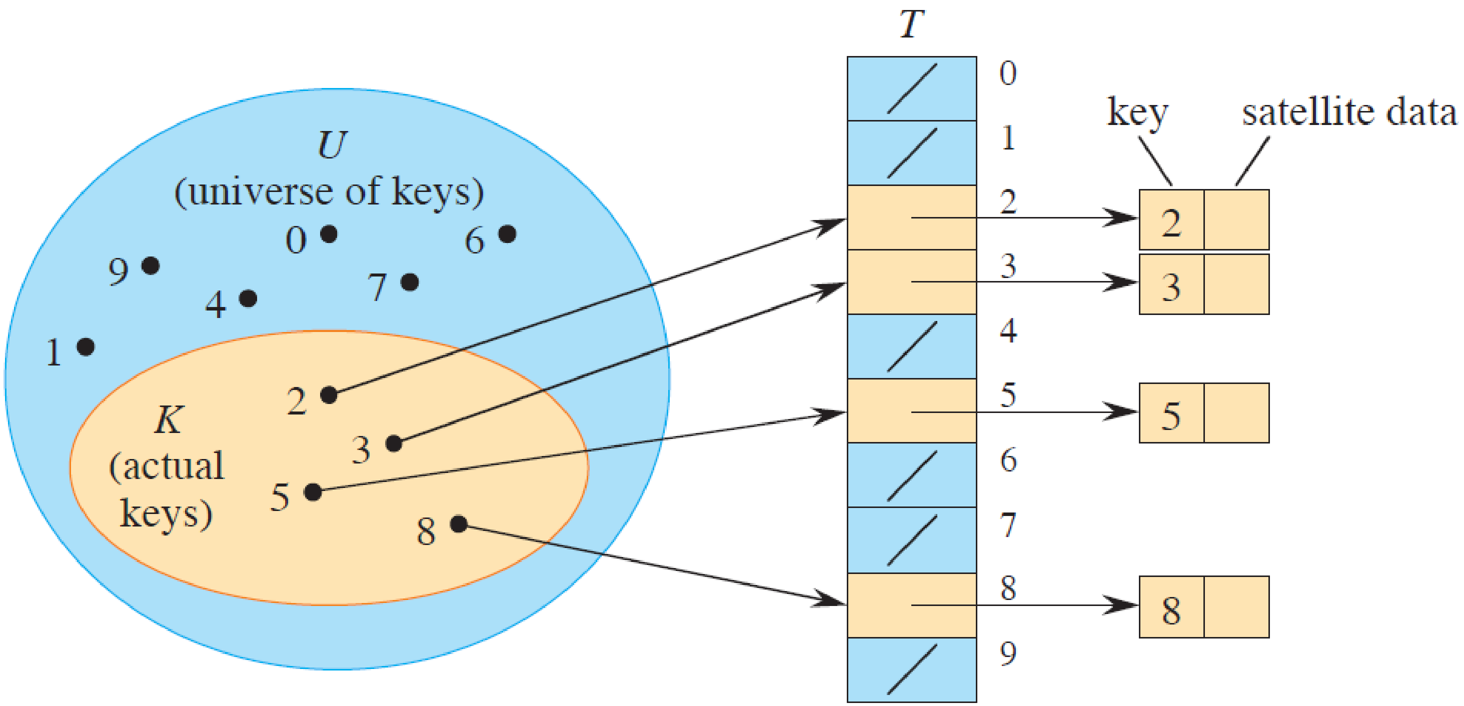
Primena “rečnika”

- Verovatno je najčešće upotrebljavana struktura podataka
- Vrlo široka primena:
  - baze podataka
  - prevodioci: imena -> promenljive
  - rutiranje mrežnog saobraćaja: IP adresa -> žica
  - virtuelna memorija: virtuelna memorija -> fizička adresa

  - pretraga podstringova (npr. *Google search*)
  - sinhronizacija sadržaja datoteka
  - kriptografija
  - …
- Implementiran u savremenim programskim jezicima

Najjednostavnija organizacija “rečnika”

- Koristi tabelu (tj. niz) sa direktnim pristupom gde je vrednost ključa indeks u tabeli



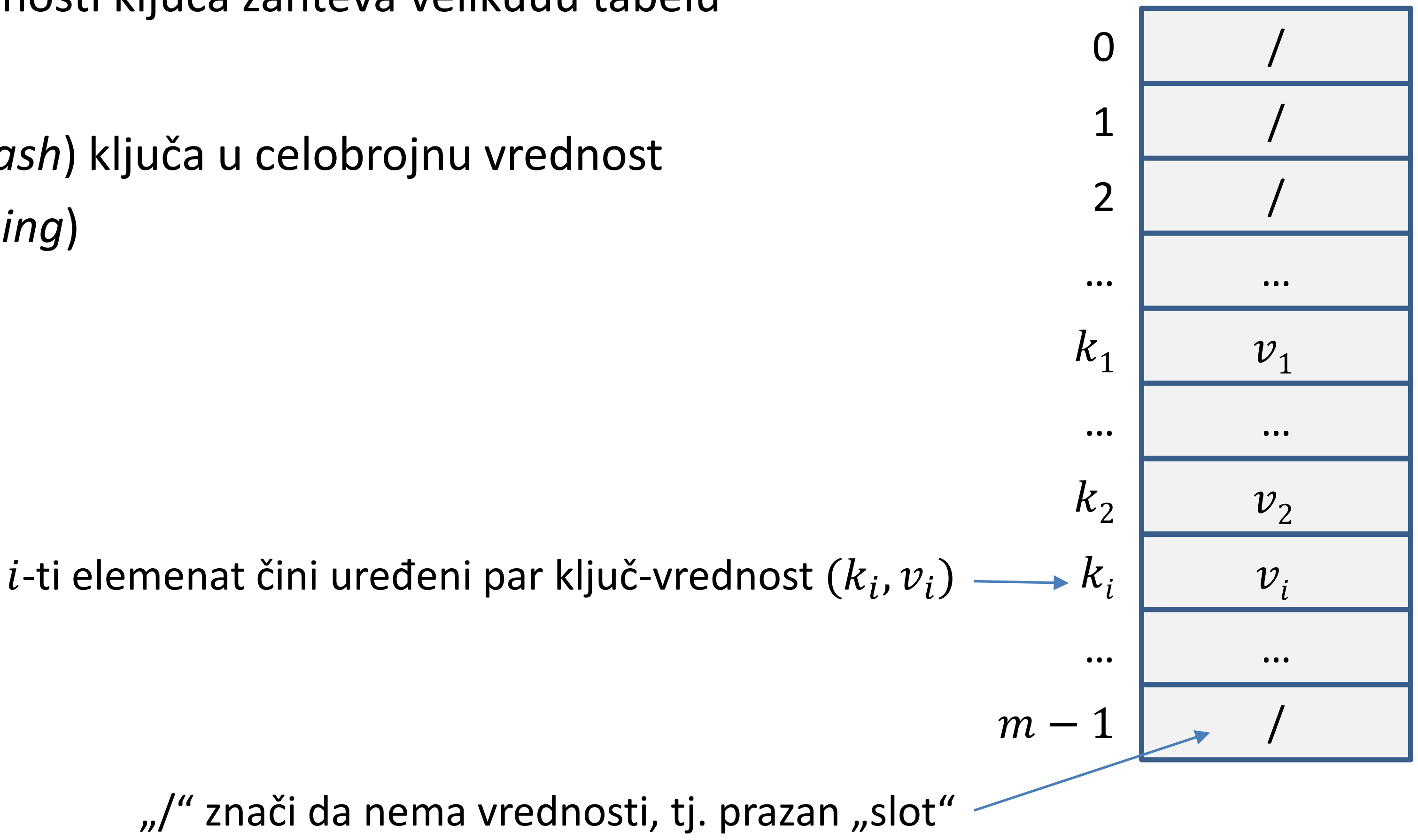

```
DIRECT-ADDRESS-SEARCH(T, k)
    return T[k]

DIRECT-ADDRESS-INSERT(T, x)
    T[x.key] = x

DIRECT-ADDRESS-DELETE(T, x)
    T[x.key] = NIL
```

Tabela sa direktnim pristupom

- Problemi:
  1. ključ mora biti nenegativna celobrojna vrednost
  2. širok opseg vrednosti ključa zahteva velikuuu tabelu
- Rešenja:
  - Za 1: prihešš (*prehash*) ključa u celobrojnu vrednost
  - Za 2: hešing (*hashing*)



Priheš

- Priheš je funkcija koja vrednost ključa prevodi u celobrojnu vrednost (nenegativnu)

$$h_p\colon k \rightarrow i$$

- Praktično, ključ se prevodi u indeks i.

- u teoriji: $x = y \Leftrightarrow h_p(x) = h_p(y)$

  $x \neq y \Leftrightarrow h_p(x) \neq h_p(y)$ u praksi: nije uvek

- Dok je elemenat u tabeli, njegova priheš funkcija h_p se ne sme menjati, jer ga onda ne možemo pronaći.

Hešing

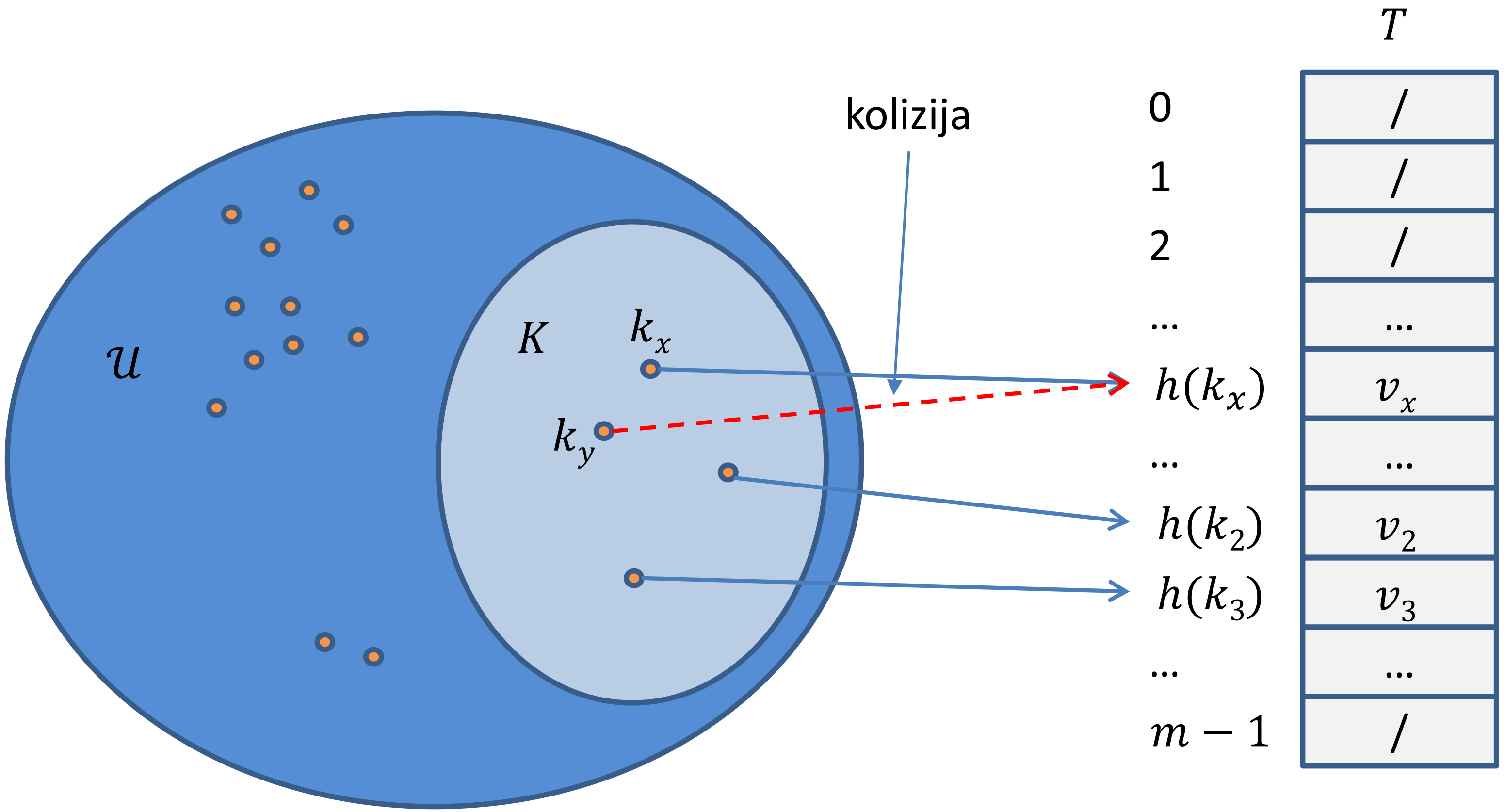
- Hešing redukuje potencijalno velike vrednosti brojeva $i = h_p(k)$ na veličinu tabele m.
- Ideja: n elemenata smestiti u tabelu T sa m redova, tj. $m \approx n$

- Heš funkcija
$$h\colon \mathcal{U} \rightarrow \{0,1,2,\ldots m-1\}$$
gde je $\mathcal{U}$ domen $h_p(k)$.

- **Problem kolizije**: za dva ključa $k_x, k_y \in K$ će se desiti kolizija ako je $h(k_x) = h(k_y)$
  - To znači da će se nakon smeštanja u tabelu para (k_x, v_x) (u red $h(k_x)$ se upiše vrednost v_k), par (k_y, v_y) ne može smestiti (nema gde) jer je red $h(k_y) = h(k_x)$ zauzet.

Hešing

- Pretpostavimo da su ključevi $\{k_1, k_2, \dots\}$ nenegativni
  (ako nisu primenimo priheš)

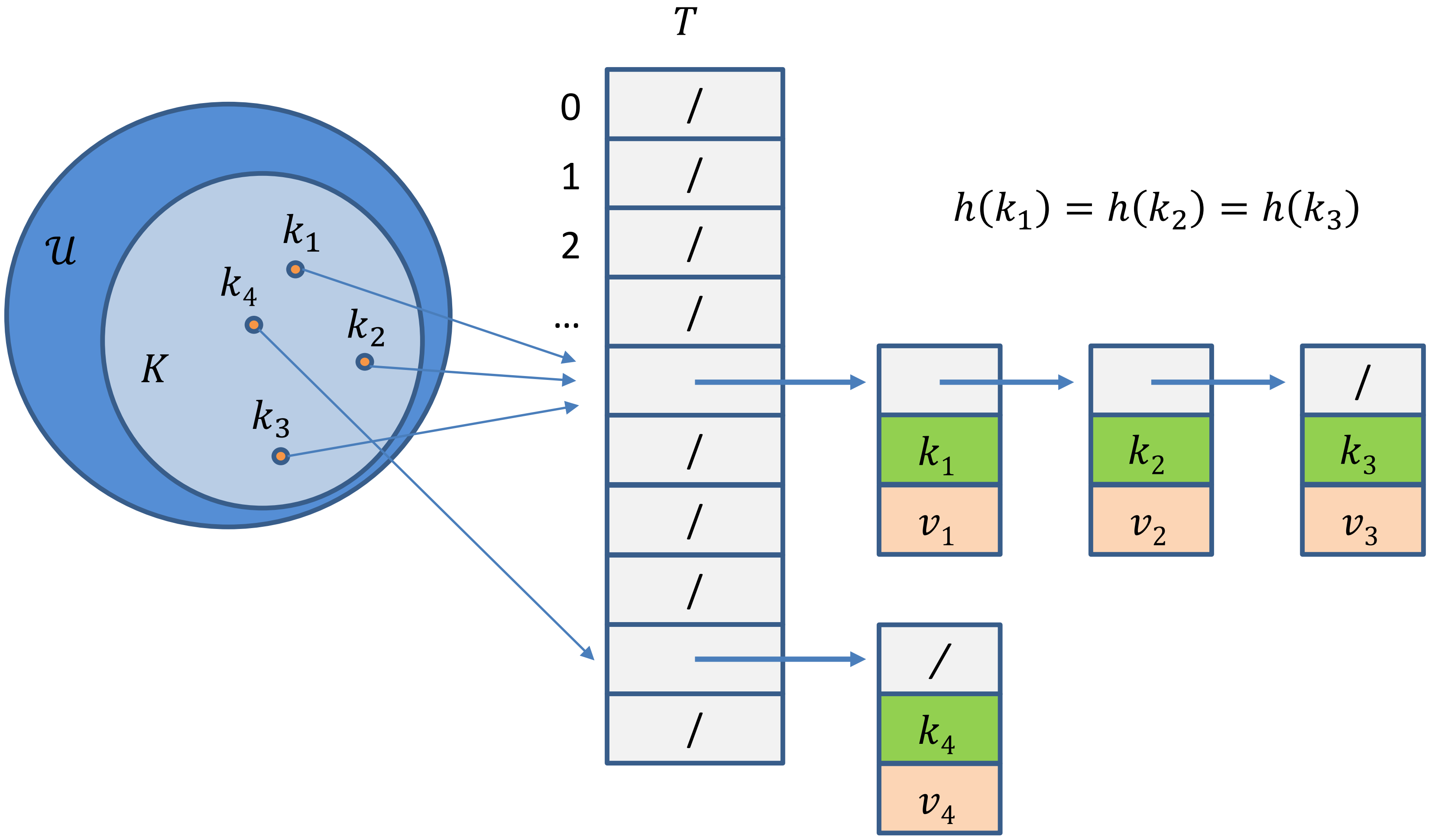


Skup vrednosti ključeva $K = \{k_1, k_2, \dots\}$ je podskup domena $\mathcal{U}$, tj. $K \subset \mathcal{U}$, ali je opseg vrednosti u K i dalje „neprijatno velik“.

Kako razrešavamo kolizije?

- Razmatramo dva rešenja:
  - Heširanje ulančavanjem (naziva se i heširanje spajanjem)
  - Otvoreno adresiranje

Hešing i ulančavanje



U listama se pamte elementi, tj. parovi ključ-vrednost (k_i, v_i). Neophodno je zapisati ključ zbog naknadne pretrage i/ili brisanja.

Heširanje ulančavanjem

- Elementi sa istom heš vrednosti se postavljaju u ulančanu listu, a tabela T se sastoji od pokazivača na liste.
- Posledica:
  - Dodavanje postavlja elemenat na početak liste. Složenost $O(1)$
  - Pretraga prolazi kroz celu listu $T[h(k_i)]$
    - Složenost $\Theta(n)$, kada se svih n elemenata nalazi u istom redu u T
  - Brisanje elementa je veoma slično pretrazi.

- Problem: kako izabrati heš funkciju da se smanje kolizije?

CHAINED-HASH-INSERT(T, x)
LIST-PREPEND$(T[h(x.key)], x)$

CHAINED-HASH-SEARCH(T, k)
return LIST-SEARCH$(T[h(k)], k)$

CHAINED-HASH-DELETE(T, x)
LIST-DELETE$(T[h(x.key)], x)$

Jednostavno uniformno heširanje

- (*Simple Uniform Hashing*)
- Pretpostavka: **svaki ključ ima podjednaku šansu da se mapira (hešira) na bilo koji red u tabeli, nezavisno od mesta gde se heširaju ostali ključevi**.

- Definiše se faktor popunjenosti tabele (*load factor*) $\alpha = \frac{n}{m}$
  - n broj elemenata koje smeštamo u T
  - m broj mesta (redova) u T
  - α određuje očekivani broj elemenata po mestu, tj. očekivana dužina ulančane liste. (očekivana vrednost je srednja vrednost)
- Performanse: očekivano trajanje pretrage je $\Theta(1 + \alpha)$
  - 1 za primenu heš funkcije i pristup redu tabele, a α za prolaz kroz listu.
  - Ako je $m = \Omega(n)$, tada je $\alpha = O(1)$, pa je i trajanje pretrage $O(1)$!!!

Dobra Heš funkcije

- Dobra heš funkcija zadovoljava/aproksimira osobinu jednostavnog uniformnog heširanja
  - svaki ključ ima podjednaku šansu da se mapira (hešira) na bilo koji od m redova u tabeli
  - Retko imamo priliku da proverimo tu osobinu jer se obično ne zna funkcija raspodele koja generiše ključeve. Takođe, možda uzastopno generisani ključevi nisu nezavisni

- Primer: ako se radi o ključevima koji su slučajno generisani realni brojevi sa uniformnom raspodelom u opsegu $0 \leq k < 1$, tada

$$h(k) = \lfloor km \rfloor$$

  zadovoljava osobine jednostavnog uniformnog heširanja.

- Kvalitativna analiza ključeva može biti korisna tokom dizajna rešenja, mada se u praksi obično upotrebljavaju heurističke (iskustvene) tehnike

Primeri Heš funkcije

- Ključevi se često predstavljaju kao **prirodni brojevi** $k \in \mathbb{N}$
  - Npr. tekst predstavljen ASCII znacima se može predstaviti kao broj
    - Primer: “danas”
      $100 * 128^4 + 97 * 128^3 + 110 * 128^2 + 97 * 128^1 + 115 * 128^0 = 27048784115$
- Stoga, heš funkcije tipično imaju kao parametar veliki prirodan broj

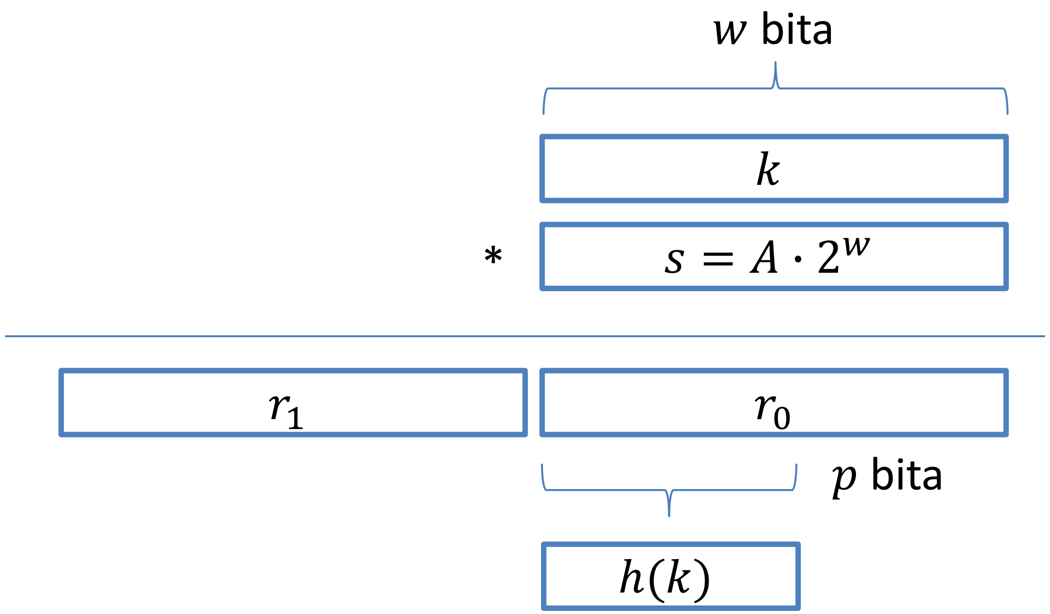
Heš funkcija – Metod deljenja

- **Metod deljenja**: $h(k) = k \bmod m$
  - Jednostavan metod, ali treba izbeći zavisnost od šablona koji postoji u vrednosti ključa
    - Npr. celobrojna vrednost ključa koja predstavlja datum oblika GGGGMMDD (kao što je 20210427) na pozicijama DD neće imati vrednost 0 veću od 31, a na poziciji MM veću od 12.
    - Posledica je da se neki indeksi nikada ne generišu, a javljaju se kolizije na drugim mestima.
  - m treba da je prost broj, ali ne blizu stepena 2 ili 10.
  - Kada je $m = 2^p$ uzima se donjih p bita vrednosti ključa

Heš funkcija – Metod množenja

- Heš funkcija je: $h(k) = \lfloor m(kA \bmod 1) \rfloor$ gde su: $0 \leq A < 1$, a $kA \bmod 1$ je deo iza decimalne tačke , tj. $kA - \lfloor kA \rfloor$
  - Time se svodi na „dobru“ heš funkciju

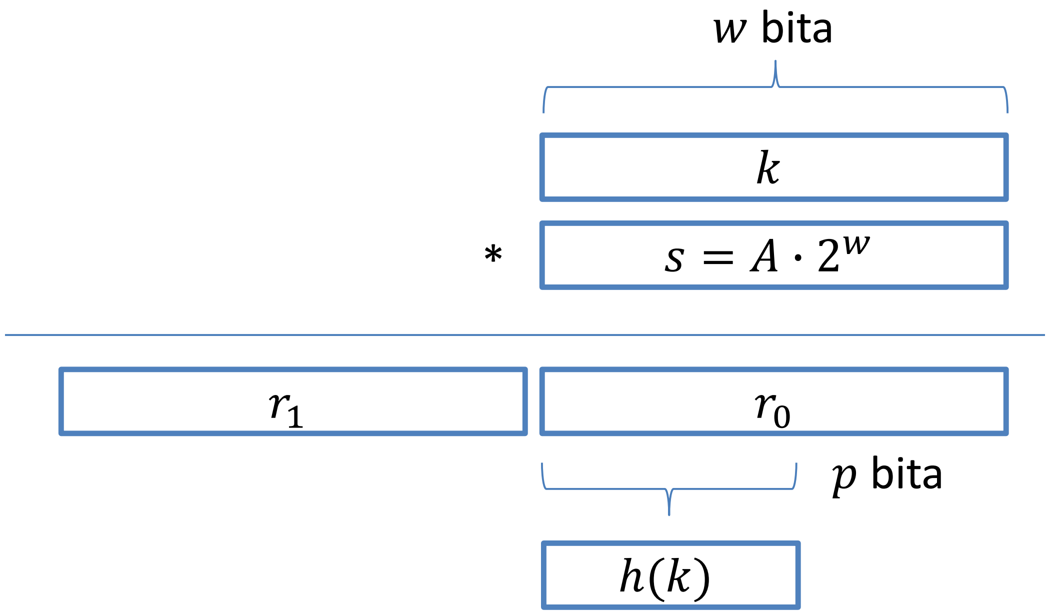
- Implementacija: bira se $m = 2^p$ i pretpostavlja se da ključ „upada“ u opseg jedne računarske reči koja ima w bita ($0 \leq k < 2^w$)
  – brzo se sprovodi
  – nije osetljiva na vrednost m



Primetiti da sve funkcije heširanja zavise od m.

Heš funkcija – Metod množenja

- Primer:



$k = 123456$

$p = 14, m = 2^{14} = 16384, w = 32$

$A = \frac{\sqrt{5}-1}{2} = 2654435769/2^{32}$

$k \cdot s = 327706022297664 = 76300 \cdot 2^{32} + 17612864$

$r_1 = 76300$

$r_0 = 17612864 = (00000001000011001100000001000000)_2$

$h(k) = (00000001000011)_2 = 67$

$$h(k) = \lfloor m(kA \bmod 1) \rfloor$$

Koliko treba da je m?

- Malo m: operacije su spore, veliko m: „bačen prostor“
- Želimo da je $m = \Theta(n)$ i kada se dodaju i brišu elementi
- Rešenje: m je promenljivo, po pokretanju malo, a može se po potrebi povećavati (smanjivati).
- Ako se poveća tabela T mora se ponoviti heširanje jer se promenilo m
  - Primenjuje se heš funkcija na sve elemente u tabeli i ona se popunjava „od nule“: trajanje operacije $\Theta(n + m) \approx \Theta(n)$ za $m = \Theta(n)$.

- Za koliko povećati T (kada je $n = m$)?
  - **Pogrešna strategija**: povećati tabelu za jedno mesto ($m \mathrel{+}= 1$) jer se ponovo računa heš vrednost za svih m elemenata u tabeli.
    Tada n dodavanja traje $\Theta(1 + 2 + \cdots + n) = \Theta(n^2)$ (polazeći od $m = 1$).
  - **Ispravna strategija**: dupliranje tabele ($m \mathrel{*}= 2$)?
    n dodavanja traje $\Theta(1 + 2 + 4 + 8 + \cdots + n) = \Theta(n)$.
    Nekoliko dodavanja traje linearno sa brojem elemenata, ali u proseku je $\Theta(1)$, tj. kaže se da je amortizovano vreme izvršavanja $\Theta(1)$

Amortizovano vreme izvršavanja

- Operacija ima amortizovano vreme izvršavanja $T(n)$ ako je za k operacija trajanje $\leq k \cdot T(n)$.
- Grubo gledano: amortizovano vreme je prosečno vreme za ponovljene operacije.

Sumarno – operacije sa heš tabelom

Trajanje operacija kada se primeni dupliranje tabele:

- Dodavanje: amortizovano vreme izvršavanja je $O(1)$.
- Pretraga: $\Theta(1)$ jer se održava $m = \Theta(n)$ i tada je α konstanta
  - α je konstanta kada se koristi jednostavno uniformno heširanje ili univerzalno heširanje.
- Brisanje: amortizovano vreme izvršavanja je $O(1)$
  - kada n padne ispod $m/4$ tabelu treba prepoloviti
  - ako se tabela prepolovi kada je n = $m/2$, a duplira kada je n = m+1, tada za n = m i niz operacija dodavanja i brisanja (sled: dodaj, obriši, dodaj , obriši, ...) dobijamo linearno vreme izvršavanja.

Otvoreno adresiranje

- (drugi) način da se reše kolizije u heš tabeli
- nema ulančanih elemenata
- najviše je jedan elemenat po redu u T, tj. mora biti ispunjeno: $m \geq n$

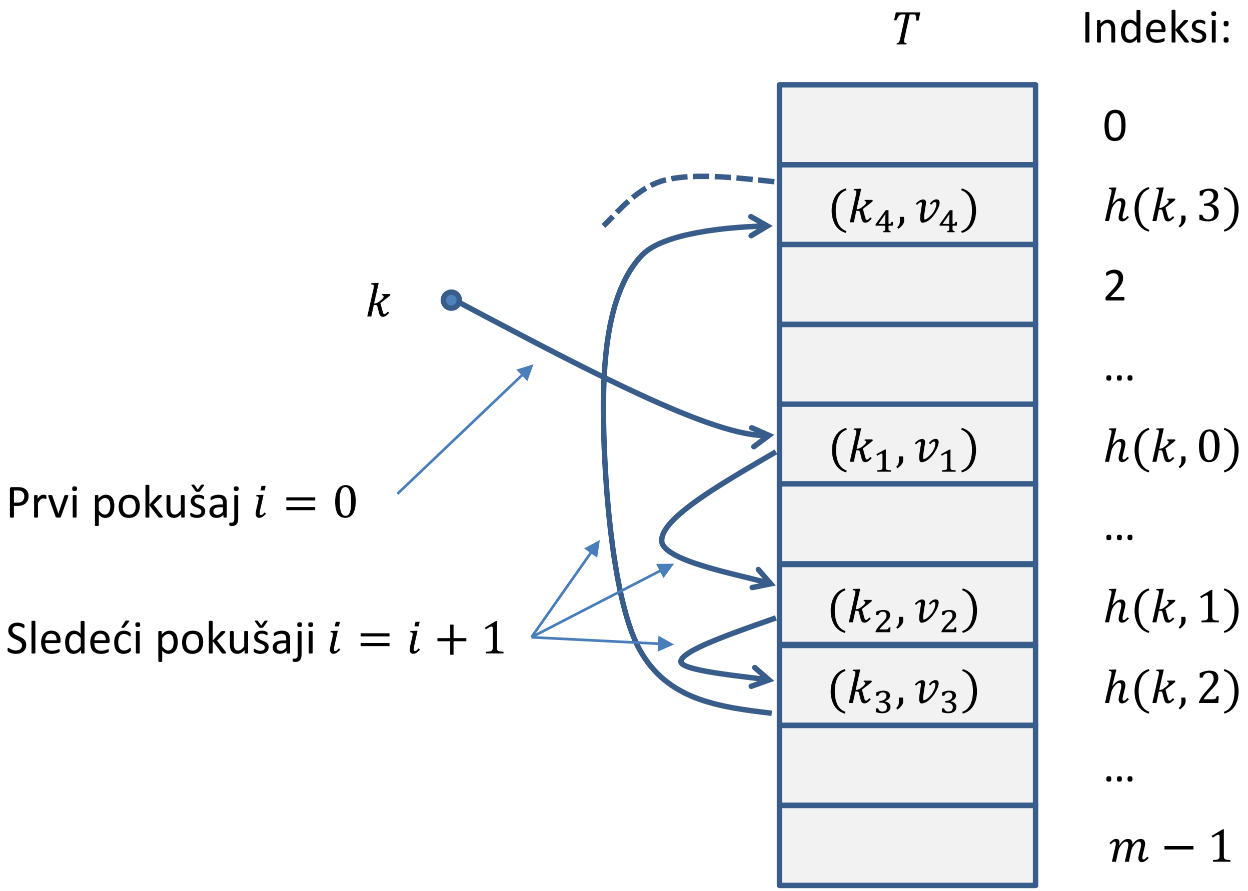
- heš funkcija je „proširena“ i pored ključa koristi i broj pokušaja heširanja
(funkcija heširanja ima dva parametra)

$$h: \mathcal{U} \times \{0,1,\dots,m-1\} \rightarrow \{0,1,\dots,m-1\}$$

  – h je funkcija koja mapira par $(ključ, pokušaj)$ na vrstu u tabeli T.
  – Ideja: ako je na dobijenom indeksu tabela popunjena parametar i se povećava $h(k,i)$ (a k se ne menja).
  Teorijski gledano, na takav način će biti adresirani svi redovi u tabeli, a proces traje dok se ne pronađe prazan „slot“.

0	/
1	/
2	/
…	…
	(k_1, v_1)
	…
	(k_3, v_3)
	(k_2, v_2)
	…
$m-1$	/

Otvoreno adresiranje i heširanje



U tabeli se pamte elementi, tj. parovi ključ-vrednost (k_i, v_i). Neophodno je zapisati ključ zbog naknadne pretrage i/ili brisanja.

Dodavanje

- Uporno probati dok se ne pronađe prazan red. Onda dodati elemenat u red.

```
DODAJ(k,v)                       // k-ključ, v-vrednost
for i = 0 to m-1
  if T[h(k,i)] == Nil            // prazan slot?
     T[h(k,i)] = (k,v)           // sačuvaj el.
     return
raise "puna tabela"              // greška
```

Pretraga

- Pokušavati primenu heš funkcije dok se u indeksiranom slotu ne nađe traženi ključ ili se ne „natrči“ na prazan slot.

```
PRETRAŽI(k)
for i = 0 to m-1
  if T[h(k,i)] == Nil                  // prazan slot?
    return Nil                         // kraj „lanca“
  elseif T[h(k,i)].ključ == k          // u slotu je ključ?
    return T[h(k,i)]                   // vrati elemenat
return Nil                             // pretražen ceo T
```

Brisanje

- Povećavanjem broja pokušaja i za istu vrednost ključa pravi „prividno ulančavanje“ (koje je posledica kolizije) i stoga se ne može jednostavno ukloniti elemenat iz T (tako što se postavi prepiše sa Nil) jer se „prekida“ zamišljeni lanac i nakon toga pretraga neće raditi.

- Rešenje: uvodi se posebna oznaka sa značenjem „obriši me“ za svaki red (slot) u tabeli T
  - dodavanje ignoriše tu oznaku (tretira je kao Nil), ali pretraga je tumači kao da je red popunjen (da se „lanac pretrage“ ne bi prekinuo)
  - kod smanjivanja tabele označeni redovi se brišu.

Strategija heširanja sa „pokušajima“

- **Linearno dodavanje pokušaja** (*linear probing*)

$$h(k,i) = (h'(k) + i) \bmod m$$

gde je $h'(k)$ „obična“ heš funkcija.

  - Problem: dovodi do zauzeća uzastopnih redova – prave se zauzeti blokovi koji vremenom postaju sve veći

- **Duplo heširanje**

$$h(k,i) = \left(h_1(k) + i \cdot h_2(k)\right) \bmod m$$

gde su $h_1(k)$ i $h_2(k)$ „obične“ heš funkcije

Poređenje otvorenog adresiranja i ulančavanja

- Otvoreno adresiranje (OA) zauzima manje memorije (ne zahteva pokazivače)
- Ulančavanje je manje osetljivo na izbor heš funkcije.
  - Kod OA treba paziti da se izbegnu blokovi zauzetih redova.

Druge primene heširanja

- Poređenje stringova: Data su dva stringa s i t. Da li se s pojavljuje u t kao podstring? (koliko puta?)
  - Jednostavan algoritam poredi s sa delovima t iste dužine. Složenost: $O(|s| \cdot |t|)$
  - Karp-Rabin algoritam koristi heš vrednosti podstringova koji se porede. Složenost: $O(|s| + |t|)$
    - Koristi se posebna heš funkcija: *Rolling Hash ADT*

- Kriptografska heš f-ja je deterministička procedura koja uzima proizvoljan blok podataka i vraća bit-string konačne dužine kao heš vrednost.
  - Ako se neki podataka u bloku promeni (slučajno ili namerno) heš vrednost mora biti drugačija